Shahin Safari

OptiAnalyser - Web-pohjainen tilausprosessin seuranta- ja analysointisovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

18.05.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Shahin Safari OptiAnalyser - Web-pohjainen tilausprosessin seuranta- ja analysointisovellus 48 sivua + 1 liite 18.5.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Lehtori Simo Silander Optiscan Oy Tekninen Johtaja Panu Lehikoinen

Opinnäytetyötä tehtiin Optiscan Oy:n toimeksiannosta, joka tarjoaa asiakkailleen Abakus-varastonhallintajärjestelmän.

Opinnäytetyölle asetettiin kaksi tavoitetta. Ensimmäisenä tavoitteena oli tutkia Angular 5, Vert.x-työkalupakki ja Kotlin-ohjelmointikieli. Toisena tavoitteena oli rakentaa näistä teknologioista graafinen web-sovellus, jonka avulla voitaisin tehostaa varastopäällikön työtä ja nähdä varaston kokonaiskuva.

Tutkimusosuudessa tutustuttiin Angular 5 -sovelluskehyksen, Vert.x-tykalupakin ja Kotlin-ohjelmointikielen ominaisuuksiin. Selvitettiin näiden teknologioiden rakenteet ja tuotiin esiin niiden hyvät ja huonot puolet.

Projektiosuutta toteutettiin ohjelmistotuotannon käytäntöjen mukaisesti. Määriteltiin web-sovelluksen ominaisuudet ja suunniteltiin sen arkkitehtuuria ja moduulien rakennetta. Toteutettiin sovelluksen komponentit ja käytiin läpi niiden algoritmit.

Työn tuloksena asetetut tavoitteet saavutettiin. Toteutettiin web-sovellus, joka täytti toimeksiantajan kaikki vaatimukset.

Avainsanat	Angular 5, Vert.x, Kotlin, TypeScript

Author	Shahin Safari
Title	OptiAnalyser – Web-Based Application for Monitoring and Analysing Order Process
Number of Pages	48 pages + 1 appendice
Date	30 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Simo Silander, Senior Lecturer Panu Lehikoinen, Technical Director at Optiscan Oy

This bachelor's thesis is created for Optiscan Oy, which is offering Abakus warehouse management system to its customers.

Two different goals were set for this study. The first objective was to study Angular 5 application platform, Vert.x tool-kit and Kotlin programming language, and the second aim was to build a graphical web application with these technologies. The application would enable warehouse managers to work more efficiently and to see warehouse situation better.

Research part was focused primarily on features of the Angular 5 application platform, Vert.x tool-kit and the Kotiln programming language. Structures of these technologies were explored in depth and their good and bad sides were revealed.

The project part was carried out in accordance with the practices of software production. The features of the web application were defined. The architecture and the structure of the modules were designed. Components of the application were implemented and their algorithms were analysed.

As the result of the work all objectives were achieved. The web application was implemented in the way that it fulfilled all the requirements of the client.

Keywords	Angular 5, Vert.x, Kotlin, TypeScript

Sisällys

Liitteet

Liite 1. OptiAnalyserin käyttötapaukset

Lyhenteet

I / O	Input ja output. Tiedon siirtämistä tai signaloimista tietokonelaitteiston komponenttien välillä.
JSON	Lyhenne sanoista JavaScript Object Notation. Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
WMS	Warehouse management system eli varastonhallintajärjestelmä.
PoC	Proof of concept eli soveltuvuusselvitys.
URL	Lyhenne sanoista Uniform Resource Locator. Sitä käytetään osoittamaan WWW-sivuja.
REST	Lyhenne sanoista Representational State Transfer. REST on HTTP-protokollaan perustuva arkkitehtuurimalli REST-rajapintojen toteuttamiseen.
HTTP	Lyhenne sanoista Hypertext Transfer Protocol. Se on protokolla, jota selaimet käyttävät tiedonsiirtoon.
HTML	Lyhenne sanoista Hypertext Markup Language. Se on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä.
DOM	Lyhenne sanoista Document Object Model. On tapa kuvata HTML-dokumentin.
UI	Lyhenne sanoista User Interface eli käyttöliittymä.
CSS	Lyhenne sanoista Cascading Style Sheets. Sillä määritellään ulkoasu HTML-kielellä kuvatulle rakenteelle.

1 Johdanto

Optiscan Oy tarjoaa asiakkailleen Abakus-ratkaisuja, jotka tehostavat kaikkia varaston toiminnan osa-alueita tavaran vastaanotosta lähettämötoimintoihin. Esimerkiksi Abakus Warehouse on yrityksen kehittämä WMS-ominaisuuksia ja toiminnallisuuksia sisältävä järjestelmä, joka on suunniteltu reaaliaikaisten varastotoimintojen hallintaan ja seurantaan.

Taustatutkimuksen perusteella päätettiin toteuttaa graafisen analysointityökalun, joka esittäisi tietokannassa olevia varastotietoja kuten tilausten tilannetta graafisesti selaimella. Työkalulla voi olla muitakin ominaisuuksia, kuten resurssianalysointia. Idea esitettiin yrityksen johdolle ja sieltä saatiin vihreää valoa.

Tällä opinnäytetyöllä on näin ollen kaksi tavoitetta. Ensimmäisenä tavoitteena on perehtyä ja tutkia, miten valitut uudet teknologiat toimivat, millainen rakenne niillä on, mitkä ovat niiden mahdolliset hyödyt tai haitat. Opinnäytetyön toisena tavoitteena on toteuttaa yrityksen tavoitteeseen sopiva sovellus, joka olisi toteutettu kaikilla näillä uusilla teknologioilla. Minimitavoitteeksi asetettiin selaimella toimiva graafinen ohjausnäkymäsovellus, joka voisi toimia ikään kuin platformina, johon voisi lisätä helposti uusia toiminnollisuuksia. Tämän opinnäytetyön laajuuden osalta hyväksyttäväksi tavoitteeksi asetettiin kaksi eri toiminnallisuutta.

2 Projektissa käytetyt teknologiat

OptiAnalyser koostuu kahdesta erillään olevista ohjelmistoprojekteista. Asiakaspuoli (client-side) toteutetaan omana projektina Angular 5 -sovelluskehyksen avulla. Vastaavasti palvelinpuoli (server side) toteutetaan omana projektina Kotlin-ohjelmointikielellä käyttäen Vert.x-työkalupakkia.

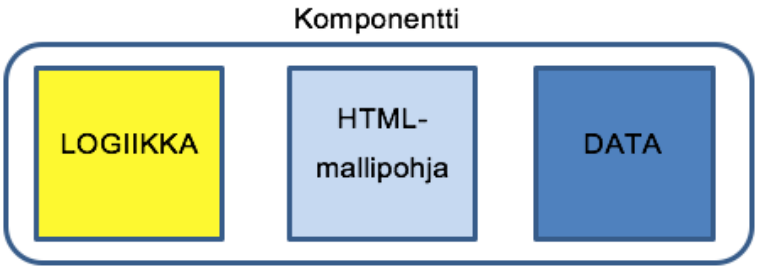
Tässä luvussa käydään läpi projektissa käytetyt pääteknologiat. Selitetään lyhyesti niiden rakenteet, ominaisuudet, toiminnallisuudet sekä hyvät ja huonot puolet. Lisäksi katsotaan projektin toteuttajan näkökulmalta näiden teknologioiden oppimiskaari ja vaatimustaso sekä niiden tukiyleisöltä saatavilla oleva tuen saatavuus ja riittävyys.

2.1 Angular-sovelluskehys

Angular on avoimeen lähdekoodiin perustuva Googlen kehittämä ja ylläpitämä web-sovelluskehys. [2.] Alun perin Googlen Angular-kehitysryhmä kehitti AngularJS-sovelluskehyksen, mutta versio 2.0 kirjoitettiin kokonaan alusta TypeScript-ohjelmointikielellä. Tämä versio on huomattavasti erilaisempi kuin aikaisempi versio ja sitä ylläpitävät Googlen lisäksi myös ulkopuoliset yritykset ja yksilöt. Google päätti, että versio 2.0 ja sitä uudempia versioita kutsutaan pelkästään Angulariksi. Samoin versiota 2.0 vanhempia versioita kutsutaan AngularJS:ksi. Näin tehtiin, koska haluttiin välttää sekaannusta kehittäjien keskellä. [2.] Tässä projektissa käytettiin Angular 5 -sovelluskehystä.

2.1.1 Angularin arkkitehtuuri

Angular-sovellukset koostuvat palveluista (services), mallipohjista (templates) ja komponenteista (components), jotka ohjaavat näkymän kenttiä. Angular omaksuu komponenttipohjaisen arkkitehtuurin, joka muistuttaa MVC-arkkitehtuuria. Jokaisessa Angular-sovelluksessa on vähintään yksi komponentti, joka on usein nimeltään ”app component”. Angularissa komponentit koostuvat kolmesta osasta (kuva 1).



Kuva 1. Angularissa komponentin rakenne.

Logiikkaosassa on koodi, jota käytetään näkymän tukemiseen. Se koostuu TypeScriptluokasta, joka sisältää tarvittavat kentät ja metodit. HTML-mallipohjassa on näkymän esittämiseen tarvittava HTML-koodi sekä mahdollinen CSS-tyylitiedosto. Dataosassa kerrotaan Angularille, miten pitäisi käsitellä komponentissa määriteltyä luokkaa. Komponentit käyttävät palveluita (services). Palvelu on tyypillisesti luokka, jolla on hyvin tarkasti määritelty tarkoitus. Jotta komponentti voisi käyttää palveluita, Angular injektoi palvelut riippuvuusinjektion avulla komponentteihin. [1.]

2.1.2 Angularin tärkeimmät ominaisuudet

Uudelleenkäytettävyys

Uudelleenkäytettävyys on Angularin yksi tärkeimmistä ominaisuuksista. Angular-sovellus koostuu monista erilaisista komponenteista, jotka ovat kuitenkin hyvin itsenäisiä. Tästä syystä ne ovat tarvittaessa helposti uudelleenkäytettävissä sovelluksen muissa osissa. Tämä on mahdollista, sillä Angularissa komponentit voivat sisältää muitakin komponentteja. Tämä on erityisen hyödyllistä silloin, kun sovelluksessa on monia samanlaisia elementtejä, kuten hakukenttiä, päivämääriä, lajittelulistoja jne.

Helppo ylläpidettävyys

Angularissa komponentit ovat hyvin kapseloituja ja itsenäisiä. Komponentteja voidaan tarvittaessa helposti irrottaa tai korvata paremmin toteutetuilla uusilla komponenteilla. Angularissa komponentit ylläpidetään erillään. Tästä syystä koodin päivittäminen, ongelmien selvittäminen ja mahdollisesti tarvittavat korjaukset helpottuvat. Komponenttien riippumattomuus yksinkertaistaa yksikkötestejä ja laadunvarmistusmenetelmiä, joilla pyritään tarkistamaan sovelluksen pienimpien osien

suorituskykyä. Toisin sanoen Angularin komponenttiarkkitehtuurin avulla web-sovelluksen ylläpidon tehokkuus kasvaa ja kustannukset vähenevät.

Korkea suorituskyky

Angular-sovelluskehyksellä toteutetuilla web-sovelluksilla on korkea suorituskyky. Tämä johtuu pääosin Angularin hierarkkisesta riippuvuusinjektointitekniikasta. Tämä tekniikka suorittaa komponentit ja niiden riippuvuudet rinnakkain irrottamalla niitä toisistaan. Angular rakentaa riippuvuuksista erillisen puun, joka voidaan muuttaa ilman, että tarvitsisi **muuttaa komponentteja, jotka käyttävät puussa olevia riippuvuuksia**. Komponentit eivät sisällä riippuvuuksia, vaan ne kuluttavat niitä ulkoisesta lähteestä. Tämä lähestymistapa antaa Angularille korkean suorituskyvyn. [4.]

Angular-sovelluksen kehitysprosessi on nopea ja tehokas. Esimerkiksi Angularin ”Universal” tuen avulla on mahdollista suorittaa Angular-sovellusta suoraan palvelimella selainten sijaan. Lisäksi Angular CLI-työkalun avulla koodimuutokset ovat nähtävissä reaaliaikaisesti selaimella, joten ei ole tarvetta jokaisen muutoksen jälkeen asentaa sovellusta uudestaan selaimelle. [3.]

TypeScript-ohjelmointikieli

Enemmistö selaimista ymmärtää ja tukee ainoastaan JavaScriptiä. JavaScript on lähes kaikkien front-end-ohjelmointiteknologioiden pohja, mutta JavaScript-koodilla on taipumus olla monimutkainen, sekava ja turhan toistuva. TypeScrptin avulla voidaan välttää näitä ongelmia. TypeScript on avoimen lähdekoodiin perustuva, Microsoftin kehittämä ohjelmointikieli. TypeScript on JavaScriptin ylijoukko (superset). Tämä tarkoittaa sitä, että TypeScript ei ole kokonaan oma kielensä, vaan se antaa JavaScriptille uusia ominaisuuksia.

TypeScriptillä koodin kirjoittaminen on nopeampaa ja puhtaampaa kuin JavaScrptillä. Puhdas JavaScript-koodin kirjoittaminen on hyvin vaativaa, vaikka kirjoittajalla olisikin paljon kokemusta. TypeScriptillä virheet huomataan jo koonnin aikana eikä ajon aikana. Näin vähän kokenutkin ohjelmoija pystyy helpommin välttämään tavallisia kirjoitusvirheitä ja huomaamaan ne. Tuloksena on puhtaammin kirjoitettu koodi, joka kääntyy päteväksi JavaScriptiksi sovelluksen koonnin aikana.

Angular on kirjoitettu TypeScriptillä, mutta kehitystyöt tehdään joko TypeScriptillä tai JavaScriptillä, sillä TypeScript käännetään JavaScriptiksi. Angular-sovellusta voidaan kirjoittaa siis sekä JavaScript-, että TypeScriptkielellä. Kuitenkin verrattuna JavaScriptiin, TypeScriptillä on lisää ominaisuuksia, jotka helpottavat ohjelmoijan työtä ja tehostavat sovelluksen kehitystä. TypeScriptillä on monista muista olio-ohjelmointikielistä esimerkiksi Javasta tutut käsitteet, jotka puuttuvat JavaScriptiltä. Näitä ovat esimerkiksi luokat, rajapinnat, enum-tietotyyppi (esimerkkikoodi 1) ja konstruktori. [7.]

```
// TypeScript
enum color {Brown = 0, Green = 1, Black = 2};
// JavaScript
var color;
(function (color) {
    color[color["Brown"] = 0] = "Brown";
    color[color["Green"] = 1] = "Green";
    color[color["Black"] = 2] = "Black";
})(color || (color = {}));;
```

Esimerkkikoodi 1. Enum-tietotyyppiesimerkki TypeScriptillä, jossa väreille annetaan omat arvonsa. Sen alapuolella nähdään, miten JavaScriptillä voidaan toteuttaa vastaava monimutkaisemmin.

2.1.3 Angularin haittapuolet

Angularin jyrkkä oppimiskäyrä

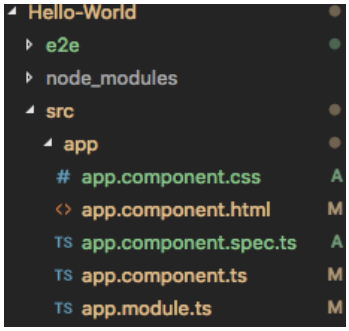
Angularissa on paljon opittavia aiheita, kuten moduulit, riippuvuudet, injektiot, komponentit ja palvelut. Uusille kehittäjille tämä tuntuu usein hyvin vaativalta. Kehittäjät, jotka ovat käyttäneet pääosiin ainoastaan JavaScriptiä front-end-kehityksessä, pitävät Angularin oppimiskäyrää vaikeana ja vaativana.

Vaikka Angular-sovellukset voidaan kirjoittaa kokonaan JavaScriptillä, kuitenkin TypeScript-kielen oppiminen ja hyödyntäminen Angular-sovelluksen kehityksessä on erittäin hyödyllistä. Tästä syystä uudet kehittäjät joutuvatkin Angularin lisäksi myös oppimaan TypeScript-kieltä, joka puolestaan vaatii aikaa ja paljon harjoitusta.

Uudet kehittäjät usein valittavat, että Angularilla on epäselviä virheilmoituksia, jotka ovat hyvin arvoituksellisia. Niiden ymmärtäminen vaatii enemmän tutkimista ja kehittäjältä kokemusta, joten aloittelijalla voi olla vaikeuksia virheiden etsimisessä ja korjaamisessa. [4.]

Angularin rakenne on monimutkainen

Angularin arkkitehtuuria pidetään monimutkaisena. Esimerkiksi vaikka aikaisemmin mainittiin, että Angularin komponenttirakenne on erittäin hyödyllinen, kuitenkin niiden hallinnointi on monimutkaista. Esimerkiksi kehittäjän täytyy luoda jopa viisi uutta tiedostoa yhtä komponenttia varten (kuva 2). Kehittäjän täytyy esimerkiksi osata komponentin luonnin lisäksi injektoida riippuvuudet oikein, jotta komponenttikokoonpano olisi toimiva.



Kuva 2. Angular-sovelluksessa app-komponentin tiedostorakenne.

2.2 Vert.x-työkalupakki

Vert.x on Eclipse-säätiön tarjoama avoimeen lähdekoodiin perustuva työkalupakki, eli se koostuu itsenäisistä kirjastoista, jotka ovat suunniteltu toimimaan yhdessä. Vert.x:llä rakennetaan reaktiivisia palvelinsovelluksia Java-virtuaalikoneelle. Reaktiivisella sovelluksella on kykyä toipua virhetilanteista hyvin sekä palvella käyttäjiä riippumatta kuormasta. Tim Fox käynnisti Vert.x-projektin vuonna 2012.

Vert.x on erittäin joustava ja nopea suorituskykyinen riippumatta siitä, onko kyseessä yksinkertainen verkkoapuohjelma tai hyvin edistynyt nykyaikainen sovellus. Vert.x-sovelluskehyksellä voidaan rakentaa suurta tapahtumavolyymia käsittävä HTTP / REST-rajapinta. Monet yritykset kuten esimerkiksi verkkokauppa Zalando käyttävät Vert.x-työkalupakkia. Lisäksi Vert.x:iä hyödynnetään myös reaaliaikaisessa pelaamisessa ja pankkien järjestelmissä. [8.]

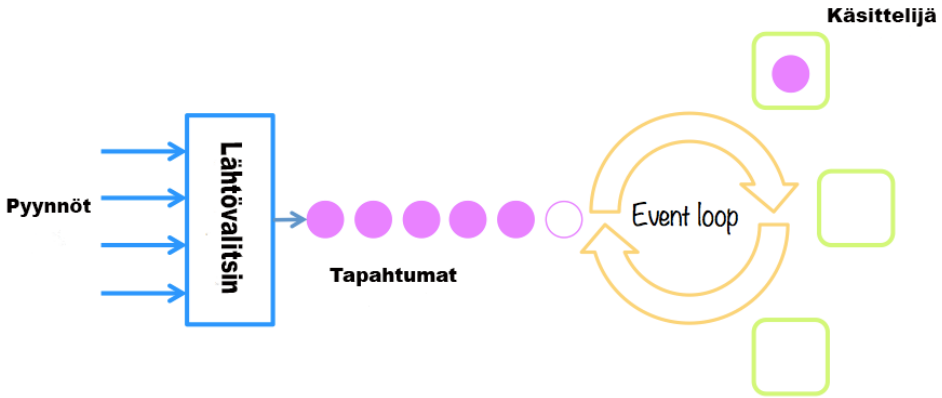
2.2.1 Vert.x-työkalupakin arkkitehtuuri

Monet verkkosovelluskehykset perustuvat yksinkertaiseen säiestrategiaan, jossa jokaiselle verkkoasiakkaalle asetetaan oma säie. Tämä säie palvelee asiakasta niin kauan kuin verkkoyhteys on olemassa. Vaikka tällä synkronisella I/O-säiemallilla on etuna olla yksinkertainen, se kuitenkin estää skaalaavuutta. Liian monet samanaikaiset yhteydet hidastavat järjestelmää ja vaativat paljon resursseja, sillä esimerkiksi käyttöjärjestelmän ydin joutuu käyttämään paljon laskentatehoa säikeiden aikatauluhallinnoitiin. Tällaisissa tapauksissa on siirryttävä asynkroniseen I/O:hon, johon Vert.x tarjoaa vankan perustan. [11.]

Kuvassa 3 nähdään, että Vert.x:n arkkitehtuuri on tapahtumapohjainen, ja se on toteutettu käsittämään paljon samanaikaisia pyyntöjä. Arkkitehtuuri on toteutettu Reactor-suunnittelumallin avulla, jossa tarkkaillaan tapahtumia keskitetysti useasta tapahtumalähteestä, ja toimitetaan ne kyseisen tapahtuman käsittelijälle.

Tapahtumasilmukka (event loop) käsittelee tapahtumajonoa purkamalla jonon tapahtumat peräkkäin käsittelijälle (handler). Näiden tapahtumien purkausprosessi on synkroninen (yksi kerrallaan). Toisaalta käsittelijät ovat asynkronisia eli epätahtisia.

On erittäin tärkeä huomata, että tapahtumasilmukka ja käsittelijät suoritetaan samassa säikeessä, joten käsittelijä ei saa koskaan estää säiettä. Jos jostakin syystä säiettä estetään esimerkiksi tietokantakyselyllä, niin se estää kaikkien muiden pyyntöjen ja tapahtumien käsittelyä. [9.]



Kuva 3. Vert.x:n tapahtumakohtainen arkkitehtuuri [9.]

2.2.2 Vert.x:n tärkeimmät ominaisuudet

Monikielisyys

Vert.x on kirjoitettu Javalla, mutta sen käyttämiseen Java-kielen osaamista ei vaadita. Itse asiassa Vert.x ei vaadi mikään tietyn ohjelmointikielen osaamista vaan se tukee useita ohjelmointikieliä. Esimerkki tuetuista kielistä ovat Java, JavaScript, Groovy, Ruby, Python, Ceylon, Scala ja Kotlin. Vert.x:n monikielisyystuen ansiosta ohjelmoija voi itse päättää, millä kielillä hän haluaisi tehdä kehitystyötä. Näin alkuun pääseminen on helpompaa ja mukavampaa, sillä kehittäjä voi valita kielen, jossa hän on vahva. Käytännössä Vert.x tarjoaa tuetuille kielille idiomaattista sovellusrajapintaa eli rajapintaa käytetään samojen käytäntöjen mukaisesti [10.] Esimerkkikoodeissa 2 ja 3 nähdään, miten Javalla ja Kotlinilla rakennetaan Vert.x HTTP-palvelinta samalla käytännöllä.

```
// import io.vertx.core.AbstractVerticle;
public class Server extends AbstractVerticle {
  public void start() {
    vertx.createHttpServer().requestHandler (req -> {
      req.response().putHeader("content-type", "text/plain")
        .end("Hello from Vert.x!");
    }).listen(8080);
  }
```

Esimerkkikoodi 2. Vert.x HTTP-palvelimen luominen Javalla.

```
// import io.vertx.core.AbstractVerticle
class Server : AbstractVerticle() {
  override fun start() {
    vertx.createHttpServer().requestHandler {req ->
        req.response().putHeader("content-type", "text/plain")
          .end("Hello from Vert.x")
      }.listen(8080)
  }
}
```

Esimerkkikoodi 3. Vert.x HTTP-palvelimen luominen Kotlinilla.

Yksinkertainen rinnakkaisuusmalli

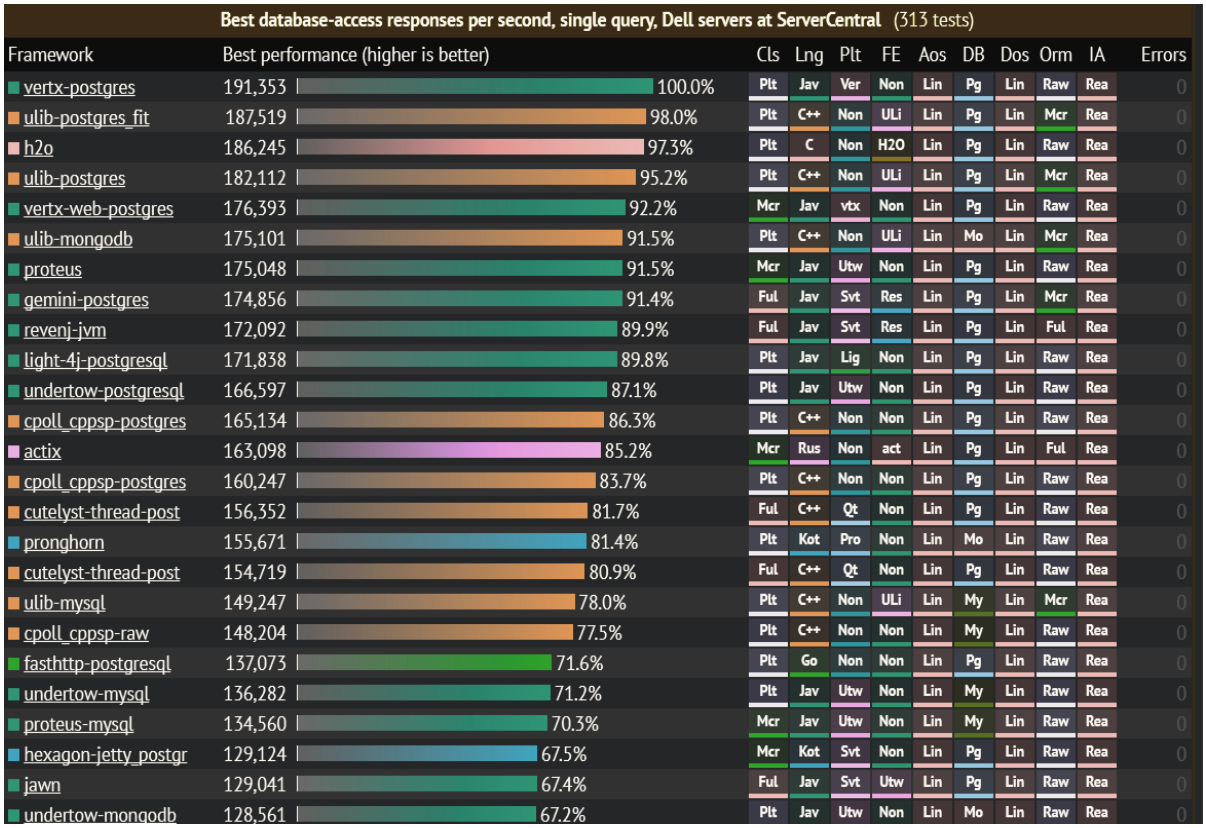
Kun palvelinsovellukset kehitetään Vert.x:llä, kehittäjä voi saavuttaa monisäikeisyyttä kirjoittamalla koodia yksisäikeisesti. Tämä onnistuu, sillä Vert.x huolehtii automaattisesti monisäikeisyydestä. Vert.x:llä on yksikertainen rinnakkaisuusmalli, jonka avulla kehittäjä voi helpommin ja nopeammin kehittää monisäikeisen palvelinsovelluksen ilman, että hänen tarvitsisi huolehtia monisäikeisyyden käsitteistä kuten synkronoinnista ja lukituksesta.

Vert.x-sovellukset rakennetaan komponenteista (verticlet), jotka toimivat itsenäisesti ja yksisäikeisesti. Näillä komponenteilla ei ole yhteistä tilaa, joten ne voivat toimia rinnakkain. Suorituskykyvaatimuksien muuttuessa näitä komponentteja voidaan helposti uudelleen järjestää vastaamaan uusia vaatimuksia.

Samaa komponenttia voidaan luoda useita kertoja, jotka voivat olla vastuussa samasta tehtävästä. Vert.x automaattisesti huolehtii tasaisen työmäärän jakamisesta niiden keskellä. Näin sovelluksen tehokkuus paranee, sillä CPU-ytimien tehoa hyödynnetään paremmin vähemmällä vaivalla. Niinpä yksinkertaisen rinnakkaisuusmallin tuloksena on helppokäyttöinen lähestymistapa tehokkaan monisäikeisen sovelluksen kirjoittamiseen, joka on myös helposti skaalattavissa. [12.]

Korkea suorituskyky

Vert.x on erittäin nopea ja kevyt. Sen ydin on kooltaan vain noin 650 kB. Kuvassa 4 on esimerkki techempower.com sivun tekemästä nopeusvertailusta.



Best database-access responses per second, single query, Dell servers at ServerCentral (313 tests)

Framework	Best performance (higher is better)	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA	Errors
vertx-postgres	191,353 100.0%	Plt	Jav	Ver	Non	Lin	Pg	Lin	Raw	Rea	0
ulib-postgres_fit	187,519 98.0%	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea	0
h2o	186,245 97.3%	Plt	C	Non	H2O	Lin	Pg	Lin	Raw	Rea	0
ulib-postgres	182,112 95.2%	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea	0
vertx-web-postgres	176,393 92.2%	Mcr	Jav	vtx	Non	Lin	Pg	Lin	Raw	Rea	0
ulib-mongodb	175,101 91.5%	Plt	C++	Non	ULi	Lin	Mo	Lin	Mcr	Rea	0
proteus	175,048 91.5%	Mcr	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea	0
gemini-postgres	174,856 91.4%	Ful	Jav	Svt	Res	Lin	Pg	Lin	Mcr	Rea	0
revenj-jvm	172,092 89.9%	Ful	Jav	Svt	Res	Lin	Pg	Lin	Ful	Rea	0
light-4j-postgresql	171,838 89.8%	Plt	Jav	Lig	Non	Lin	Pg	Lin	Raw	Rea	0
undertow-postgresql	166,597 87.1%	Plt	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea	0
cpoll_cppsp-postgres	165,134 86.3%	Plt	C++	Non	Non	Lin	Pg	Lin	Raw	Rea	0
actix	163,098 85.2%	Mcr	Rus	Non	act	Lin	Pg	Lin	Ful	Rea	0
cpoll_cppsp-postgres	160,247 83.7%	Plt	C++	Non	Non	Lin	Pg	Lin	Raw	Rea	0
cutelyst-thread-post	156,352 81.7%	Ful	C++	Qt	Non	Lin	Pg	Lin	Raw	Rea	0
pronghorn	155,671 81.4%	Plt	Kot	Pro	Non	Lin	Mo	Lin	Raw	Rea	0
cutelyst-thread-post	154,719 80.9%	Ful	C++	Qt	Non	Lin	Pg	Lin	Raw	Rea	0
ulib-mysql	149,247 78.0%	Plt	C++	Non	ULi	Lin	My	Lin	Mcr	Rea	0
cpoll_cppsp-raw	148,204 77.5%	Plt	C++	Non	Non	Lin	My	Lin	Raw	Rea	0
fasthttp-postgresql	137,073 71.6%	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea	0
undertow-mysql	136,282 71.2%	Plt	Jav	Utw	Non	Lin	My	Lin	Raw	Rea	0
proteus-mysql	134,560 70.3%	Mcr	Jav	Utw	Non	Lin	My	Lin	Raw	Rea	0
hexagon-jetty_postgr	129,124 67.5%	Mcr	Kot	Svt	Non	Lin	Pg	Lin	Raw	Rea	0
jawn	129,041 67.4%	Ful	Jav	Svt	Utw	Lin	Pg	Lin	Raw	Rea	0
undertow-mongodb	128,561 67.2%	Plt	Jav	Utw	Non	Lin	Mo	Lin	Raw	Rea	0

Kuva 4. Vert.x:n postgres clientin suorituskyky tietokantakyselyn tekemisessä. [13.]

2.2.3 Vert.x:n rakenne

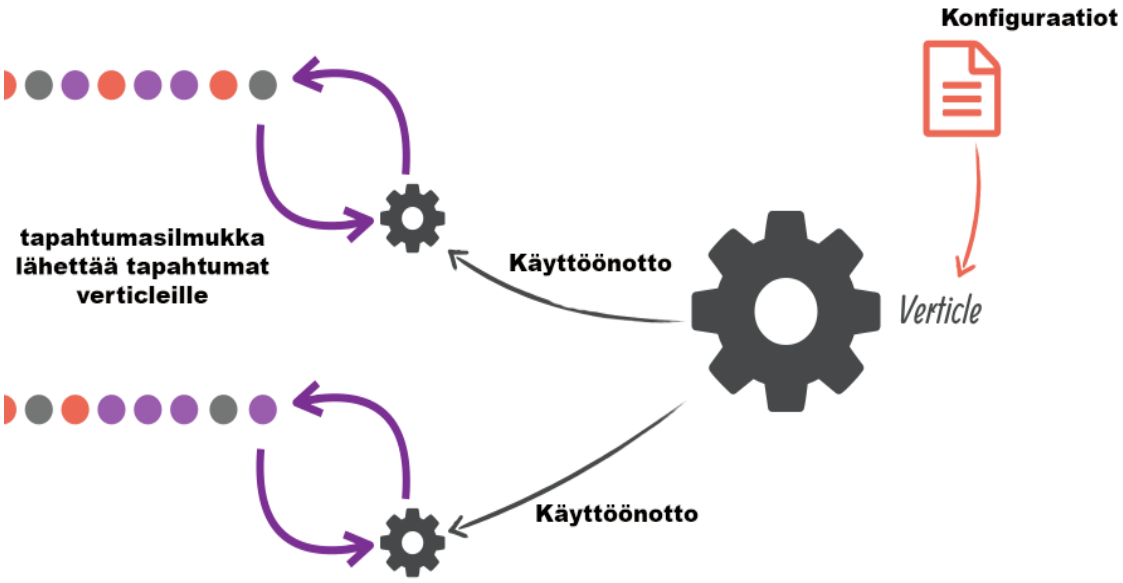
Vert.x ei ole sovelluskehys vaan työkalupakki. Sen ydinkirjasto määrittelee perusrajapinnat asynkronisen sovelluksen kirjoittamiseen, mutta kehittäjä itse valitsee tarvittavia lisämoduuleja sovellukselle. Näitä moduuleja ovat esimerkiksi tietokantayhteysmoduuli, monitorointi, lokitus, todennus jne. Vert.x ei vaadi mitään erillistä kehitysympäristöä, koska sen ydin on tavallinen jar-tiedosto. Sitä voidaan upottaa sovelluksiin yhtenä jar-tiedostona.

Vert.x-rakenteessa on kaksi avainkäsitettä, jotka ovat verticle ja event-bus. Vert.x:stä kiinnostunut ohjelmoijan täytyy ensimmäisenä oppia, mikä on verticle ja miten event-busin avulla verticle keskustelee toiselle verticleille. [11.]

Mikä on verticle?

Vert.x:n käyttöönottoyksikköä kutsutaan vertic leksi. Verticle käsittelee tapahtumasilmukasta tulevia tapahtumia. Tapahtuma voi olla esimerkiksi ajoitustapahtuma tai jonkun muun verticlen lähettämä viesti.

Jokaisella ydinsäikeellä on vähintään yksi tapahtumasilmukka. Oletuksena Vert.x asettaa kaksi tapahtumasilmukkaa CPU:n ydinsäiettä kohti (esimerkiksi jos CPU:lla on neljä ydintä, niin tapahtumasilmukoita on yhteensä kahdeksan kappaletta). Tästä johtuen tavallinen verticle prosessoi aina saman säikeen tapahtumia. Verticlet voidaan konfiguroida erikseen. Näitä konfiguraatioita ovat esimerkiksi verkko-osoitteet tai todennustiedot. Jokaista verticlea voidaan käyttöönottaa (deploy) useita kertoja. [11.] Kuvassa 5 nähdään, että verticle luodaan konfiguraatiolla. Sen jälkeen sitä voidaan ottaa käyttöön kahtena ilmentymänä. Vert.x huolehti automaattisesti pyyntöjen jaosta näiden välillä.



Kuva 5. Havaintokuva verticlen käyttöönotosta. [11.]

Mikä on EventBus?

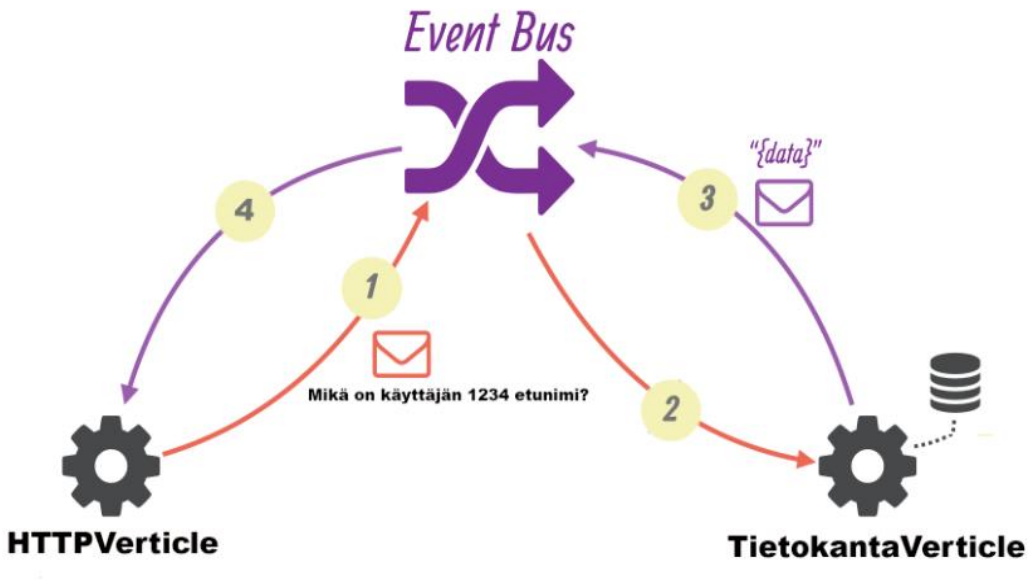
EventBus on työkalu, jolla verticle kommunikoi asynkronisesti toisen verticlen kanssa (esimerkkikoodi 4). Se sallii minkä tahansa dataformaatin siirtämisen verticlestä toiseen. Kuitenkin JSON on suositeltu dataformaatti, koska se sallii eri kielillä kirjoitettujen verticleiden kommunikointia.

```
// Tässä esimerkissä pyyntö sisältää id:n.
JsonObject request = new JsonObject().put("id", id);
// Delivery options allow us to specify headers, payload codecs and timeouts.
DeliveryOptions options = new DeliveryOptions().addHeader("action", "print");

vertx.eventBus().send("address", request, options, reply -> {
    if (reply.succeeded()) {
      context.response().setStatusCode(200);
      context.response().end();
    } else {context.fail(reply.cause());}
```

Esimerkkikoodi 4. Pyynnön lähettäminen ”action” headerillä event-busilla. Huom. Viestillä ei ole vastaanottajaa, ainoastaan lähetysosoite (address).

Kuvassa 6 nähdään esimerkkitilanne, jossa on olemassa kaksi verticleä. Näistä toinen on vastuussa HTTP-pyyntöjen käsittelemisestä ja toinen tietokannasta ja siihen liittyvistä operaatioista.



Kuva 6. Havaintokuva, jossa event-bus välittää verticleiden viestit toiselle. [11.]

HTTPVerticle käsittelee asiakkaan lähettämää viestiä ja event-busin avulla pyytää tietokantaVerticleä tekemään pyyntöön liittyvät tietokantakyselyt. Tämän jälkeen, kun tietokanta-verticle on saanut tulokset valmiiksi, tämä välittää tulokset EventBusin kautta HTTPVerticlelle. Tässä on hyvä huomata, että HTTPVerticle ei jää odottamaan tietokantaVerticlen tulosta. Lisäksi kummatkaan verticlet eivät tunne toisiaan, vaan ne tietävät osoitteen, johon viestejä lähetetään ja josta niitä luetaan. [11.]

2.3 Kotlin-ohjelmointikieli

Kotlin on monikäyttöinen, avoimeen lähdekoodiin perustuva pragmaattinen ohjelmointikieli. Kotlin on staattisesti tyypitetty ohjelmointikieli, joka yhdistää olio-ohjelmoinnin ja funktionaalisen ohjelmoinnin ominaisuuksia. Se keskittyy yhteentoimivuuteen Javan kanssa, turvallisuuteen, selkeyteen ja työkalujen tukemiseen. Kotlin toimii JVM:llä, mutta sitä voidaan myös kääntää JavaScriptiksi. Kotlinin suosio lähti kasvuun sen jälkeen, kun Google hyväksyi sen viralliseksi kieleksi Javan sijaan.Kotlinin kehitys alkoi vuonna 2010 JetBrainsin Pietarissa olevasta kehitysryhmästä. Jet Brains tunnetaan IntelliJ IDEA -ohjelmointiympäristön kehittäjänä. Jet Brains käyttää Kotlinia monissa tuotteissaan kuten IntelliJ IDEA:ssa. Kotlin on ollut vuodesta 2012 lähtien avoimeen lähdekoodiin perustuva. [14.]

Kotlin on turvallisempi ja suppeampi kuin Java. Kotlin on suunniteltu eliminoimaan null-osoittimien vaara ja tehostamaan null-arvojen käsittelyä. Kotlin tekee tätä tekemällä null-arvot laittomaksi vakiotyypeille kuten stringille ja intille (esimerkkikoodi 5). Silloin kun null-arvon käyttäminen on välttämätöntä, on käytettävä null-tyyppejä. Kotlinissa null arvo voidaan asettaa ainoastaan erityisille null-tyypeille. Nämä ovat vakiotyypit, joiden perään on lisätty kysymysmerkki (esimerkkikoodi 6). [14.]

```
var merkkijono: String = "merkkijono"

merkkijono = null // compilation error
```

Esimerkkikoodi 5. Kotlinissa Null-arvon asettaminen vakiotyypeille aiheuttaa käännösvirheen. Tässä on hyvä huomata, että Kotlinissa lauseiden ja ilmausten loppuun ei tarvita puolipistettä.

```
var merkkijono: String? = "merkkijono" // Huomio kysymysmerkkiä

merkkijono = null // OK
```

Esimerkkikoodi 6. Null-arvojen asettamiseen tarvitaan null-tyyppi, joka on muodossa vakiotyyppi+?.

Kotlin helpottaa null-arvojen käsittelyä tarjoamalla oikopolkuja, joilla voidaan välttää turhan pitkän koodin kirjoittamista. Katsotaan esimerkkiä, miten Javassa ja Kotlinissa voidaan tarkistaa, onko merkkijono null vai ei (esimerkkikoodi 7).

```
// JAVA

String abc = null;

Integer length = null;

If (abc != null){ length = abc.lenght(); } else{ length = null; }

// KOTLIN

abc: String?

length: Int?

length = abc?.lenght
```

Esimerkkikoodi 7. Kotlinilla null-arvojen käsittely on helpompaa ja suppeampaa. Tämä on erityisen kätevä tilanteissa, joissa on tarkistettava peräkkäin toisiinsa liittyviä muuttujia.

Funktionaalinen ohjelmointi

Kotlinilla on kaikki funktionaalisen ohjelmointikielen ominaisuudet ja edut. Funktionaalinen ohjelmointi on ohjelmointiparadigma, joka perustuu puhtaiden funktioiden kirjoittamiseen, yhteisen tilan ja sivuvaikutusten välttämiseen. Puhdas funktio on funktio, jonka tulos riippuu vain parametreista, eli sen arvo on aina sama samoilla parametreilla. Koska funktion tulos on aina sama, samalla parametrilla, funktiolla ei ole sivuvaikutuksia. [15.] Tarkastellaan esimerkkikoodissa 8 listan suodattamisesta Kotlinissa käyttäen anonyymifunktiota (lambda).

```
// Esimerkkilista, jossa negatiivisia ja positiivisia numeroita

val numerot = listOf(3, 2, 1, 0, -1, -2, -3)

val negatiiviset = list.filter {x -> x < 0} // -1, -2, -3
```

Esimerkkikoodi 8. Listan suodattaminen Kotlinissa käyttäen anonyymifunktiota.

Kotlin on Androidin virallinen kieli

Java ja C++ olivat Androidin virallisesti tuettuja kieliä, mutta toukokuussa 2017 Google ilmoitti, että Kotlin on Androidin uusi ja ainoa virallinen kieli. Kotlin on rakennettu Android SDK:n ja Android-studioon sisään, joka on Androidin virallinen ohjelmointiympäristö.

Kotlin käännetään samaan tavukodiin kuin Java. Kotlin on Javan kanssa yhteen toimiva. Se toimii luonnollisesti Java-kirjastojen kanssa ja jakaa Javan kanssa samoja työkaluja. Kotlin on kuitenkin yksinkertaisempi, virheettömämpi ja turvallisempi kuin Java ja takaa tehokkaampaa ja turvallisempaa Android-kehitystä. [16.]

3 Sovellusprojektiosuuden määrittely ja toteutus

Tässä luvussa esitetään tämän opinnäytetyön toisen tavoitteen eli OptiAnalyser-sovellusprojektin määrittelyä ja toteutusta.

3.1 Määrittely

3.1.1 Yleiskuva

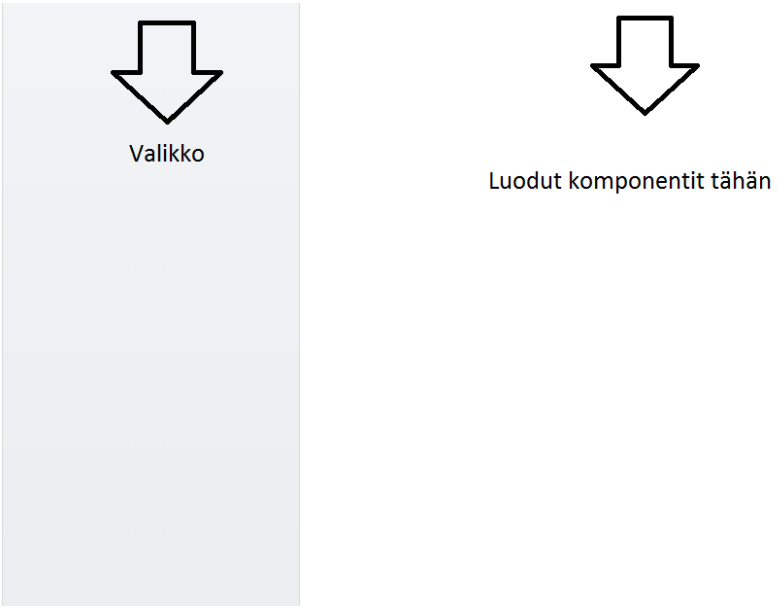
Optiscanissa pyritään jatkuvasti kehittämään ratkaisuja, joilla voittaisin tehostaa varastotoiminnan osa-alueita. Päätettiin toteuttaa graafisen, responsiivisen ja selaimella toimivan Web-sovelluksen. Sen avulla voittaisin nähdä graafisesti varaston kokonaiskuva.

Sovelluksen nimeksi valittiin OptiAnalyser. Sen avulla halutaan ensisijaisesti helpottaa varastopäälliköiden työtä, jotta varaston hallinnointi tehostuisi. Sovelluksen avulla he pystyvät hakemaan haluamansa informaatiota. Käyttäjät pystyvät sovelluksen dialogikomponentilla ja sen parametreilla rakentamaan erilaisia diagrammeja, joilla he saavat selkeän kuvan varaston tilanteesta.

Varastopäällikön lisäksi sovellusta voi käyttää kuka vain valtuutettu käyttäjä kuten muut varastontyöntekijät tai yrityksen johto. Tästä johtuen sovelluksen käytön tulisi olla mahdollisemman yksinkertaista eikä sen käyttöön pitäisi vaatia teknistä osaamista kuten tietokantakyselykielen käyttöä.

3.1.2 Käyttöliittymä ja sovelluksen sivut

Taustatutkimuksen ja asiakasvaatimusten perusteella sovelluksen käyttöliittymän ulkoasu toteutetaan ohjausnäkymäksi (dashboard), jossa näyttö jaetaan kahteen osaan (kuva 7).



Kuva 7. Sovelluksen näytönjako.

Ensimmäisessä osassa on valikko, jossa on komponenttien luontinappuloita sekä sovelluksen muut mahdolliset toiminnollisuudet kuten uloskirjautuminen ja dashboardin tallentaminen. Käyttöliittymän toisessa osassa esitetään dynaamisesti luodut komponentit. Jokainen komponentti sijoitetaan omalle erilliselle näkymälle (view), jonka avulla komponentti voidaan poistaa, sitä voidaan muokata, siirtää tai sen kokoa muuttaa sopivaan muotoon.

Diagrammeja voidaan muokata muokkausnappulan avulla, joka sijaitsee jokaisella diagramminäkymällä. Kun käyttäjä painaa muokkausnappulaa, avautuu diagrammin luontia varten käytetty dialoginäkymä, jossa käyttäjän aikaisemmat valinnat ovat näkyvissä. Käyttäjä muokkaa diagrammia muuttamalla dialoginäkymän parametreja. Tämän jälkeen diagrammi päivitetään uusien parametrien mukaisesti. Jos käyttäjä muokkaa diagrammia, joka on tallennettu selaimen muistiin, niin muokkauksen jälkeen dashboardi on tallennettava uudestaan.

Dashboardin sivulle pääsy tapahtuu kirjautumissivun avulla, jossa käyttäjältä vaaditaan tunnus ja salasana. Dashboard-sivulla ja kirjautumissivulla on oma URL-osoite. Jos käyttäjä kirjoittaa selaimen osoitekentälle suoraan dashboard-sivun osoitteen, niin hänet ohjataan kirjautumissivulle. Puolestaan jos käyttäjä on jo kirjautunut, häntä ei enää ohjataan kirjautumissivulle. Kuitenkin jos käyttäjä on jo kirjautunut ja hän kirjoittaa selaimelle kirjautumissivun osoite, niin hänet ohjataan suoraan dashboard-sivulle. Käyttäjä ohjataan virhesivulle, jos käyttäjä kirjoittaa sovelluksen verkkotunnusta sisältävän virheellisen URL:n, jota sovellus ei tunnista.

3.1.3 Toteutettavat komponentit ja niiden ominaisuudet

OptiAnalyserin tässä versioissa on kaksi dialoginäkymää, joilla rakennetaan graafisia diagrammeja. Dialogissa esitetään käyttäjälle diagrammin luontia varten saatavilla olevat parametrit. Parametrien avulla käyttäjä pystyy rakentamaan oman tarpeen mukaan räätälöityjä diagrammeja eli käytännössä samalla dialogikomponentilla pystyy rakentamaan erinäköisiä ja erisisältöisiä diagrammeja. Diagrammille voidaan antaa omaa nimi.

Ensimmäisen dialogikomponentin avulla rakennetaan diagrammeja, joiden avulla käyttäjä pystyy tarkastamaan varastotilausten tilannetta. Dialogikomponentilla käyttäjä pystyy selvittämään, kuinka monta tilausta on olemassa, missä statuksessa ne ovat tai minkä tyyppisiä ne ovat sekä mihin alueisiin ne kuuluvat. Diagrammin sisältöä voidaan rajata tietyn toimitusaikaväliin. Lisäksi tiedot voidaan hakea joko tilausten määrän tai tilausrivien mukaan.

Toisella dialogikomponentilla rakennetaan sellaisia diagrammeja, joiden avulla käyttäjä pystyy tarkastelemaan tietyn henkilön tai henkilöiden keräyshistorian. Komponentin avulla käyttäjä pystyy selvittämään, kuinka monta ja minkä tyyppisiä tilauksia on kerätty ja missä statuksessa tilaukset olivat. Sisältöä voidaan rajata tiettyyn keräysaikaväliin. Lisäksi keräyshistoria voidaan hakea joko kerättyjen tilausten määrän tai kerättyjen tilausrivien mukaan.

Luotujen diagrammikomponenttien sisältöä päivitetään automaattisesti tietyn ajan välein. Aikaväliä voidaan lyhentää tai kasvattaa suorituskyvyn mukaan, sillä päivityksessä diagrammin sisältöä pyydetään uudestaan palvelimelta, ja palvelin joutuu tekemään mahdollisesti raskaita tietokantakyselyjä. Tämä johtuu myös siitä, että diagrammin

sisältöä ei tallenneta selaimen muistiin. Jos käyttäjä tallentaa dashboardinsa selaimelle, niin muistiin tallennetaan ainoastaan dashboardissa olevien diagrammien parametrit ja koordinaatit.

Diagrammikomponentin luontivaiheessa sovellus automaattisesti määrittää diagrammille värejä riippuen siitä, kuinka paljon dataa palvelimelta on saatu. Värejä käytetään diagrammin sisällön visualisoinnissa. Sovelluksen tässä versiossa käyttäjä ei pysty määrittämään eikä muuttamaan diagrammin värejä. Jos dashboardi tallennetaan, niin tallentuu myös diagrammin värit. Tallennettu dashboardi ladataan automaattisesti silloin, kun selaimen sivua päivitetään tai kun käyttäjää kirjautuu sisään. Käyttäjä pystyy poistamaan dashboardinsa selaimen muistilta painamalla valikossa sijaitseva poista dashboard -nappulaa. OptiAnalyserin ensimmäisessä versioissa käyttäjän dashboardi tallennetaan ainoastaan selaimeen muistiin, joten selaimen muistin tyhjentäminen poistaa myös kaikkien käyttäjien tallennetut dashboardit.

3.1.4 Sovelluksen tietoturva

OptiAnalyserin käyttäjältä vaaditaan tunnus ja salasana. Tunnukset luo OptiAnalyserin ylläpitäjä. Hän luo tunnukset WMS:n tietokannan avulla, sillä tässä versiossa OptiAnalyser käyttää ainoastaan WMS:n tietokantaa eikä sillä ole omaa erillistä tietokantaa. Käyttäjän dashboardi tallennetaan hänen tunnuksen avulla, joten käyttäjät eivät pystyy näkemään toistensa dashboardia.

3.1.5 Sovelluksen ominaisuuslista

Taulukossa 1 on esitetty OptiAnalyserin ominaisuuslista, joka on pääosin kuvaus sovelluksen päätoimijasta eli varastopäälliköstä. Se kertoo, mitä hän haluaa tehdä sovelluksella ja miksi.

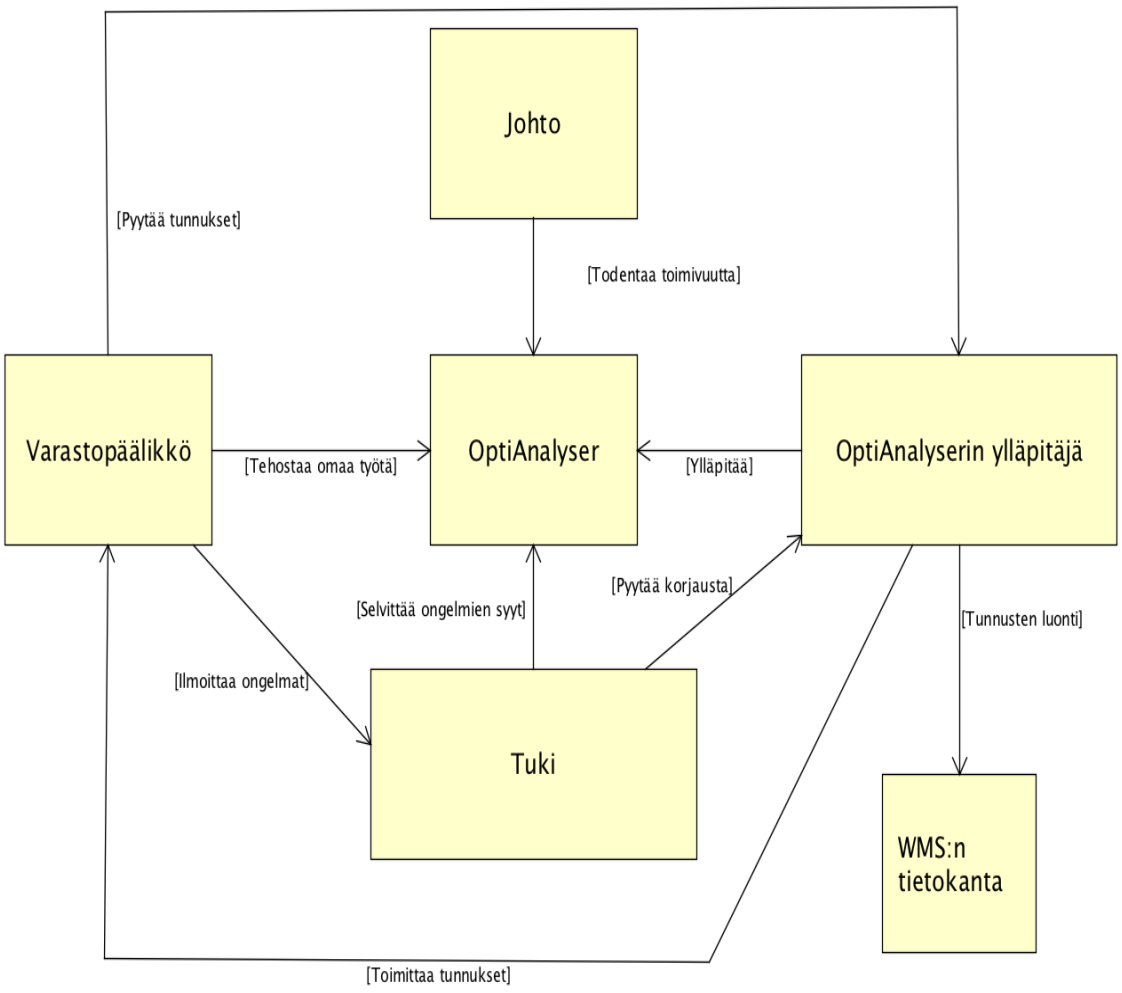
Kuka	Mitä	Miksi
Varastopäällikkönä haluan	Käyttää omaa tunnusta ja salasanaa.	jotta voisin kirjautua sisään ja käyttää sovellusta.
	Nähdä erilaiset varastotehtävät graafisesti visualisoiduissa diagrammeissa.	koska haluan hahmottaa varastotehtävien tilannetta paremmin.
	Nähdä varastotyöntekijöiden keräyshistoriaa.	koska haluan ymmärtää ja suunnitella henkilöstöresurssit paremmin.
	Luoda varastotehtävien diagrammit siihen tarkoitetulla dialoginäkymällä.	jotta voisin luoda varastotehtävädiagrammit dialoginäkymältä valitsemillani parametreilla.
	Luoda käyttäjien keräyshistorian diagrammit siihen tarkoitetulla dialoginäkymällä.	jotta voisin luoda erilaisia keräyshistoriadiagrammeja dialoginäkymältä valitsemillani parametreilla.
	Muuttaa luotujen diagrammien sijaintia tai näkymä kokoa.	jotta voisin rakentaa oman dashboardin sellaiseksi kuin haluan.
	Muokkaa luodut diagrammit.	jotta voisin muuttaa diagrammin sisältöä sellaiseksi kuin haluan.

	Poistaa luodut diagrammit.	jotta voisin tarvittaessa poistaa dashbordilta diagrammeja, joita en enää tarvitsee.
	Tallentaa oman dashboardin selaimen muistiin.	koska haluan, että dashboardini säilyisi, jos kirjaudun ulos tai jos selaimen sivua päivitän.

Taulukko 1. OptiAnalyserin ominaisuuslista

3.1.6 OptiAnalyserin sidosryhmät

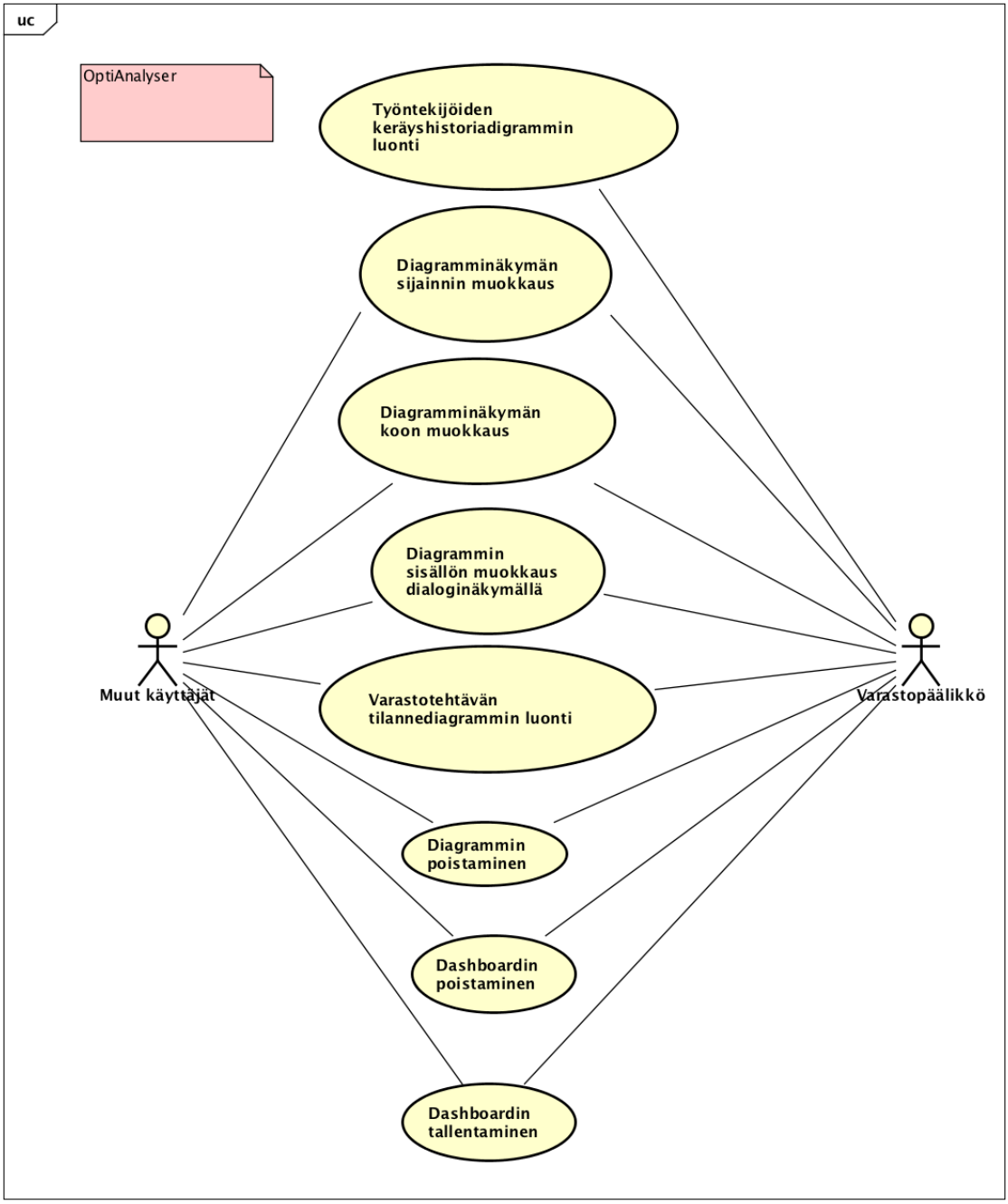
Kuvassa 8 nähdään OptiAnalyserin sidosryhmäkaavio. Kuvassa näkyvät kaikki tahot, jotka ovat OptiAnalyserin käyttöönoton jälkeen sen kanssa tekemisissä. Kuvan avulla selviää, mitä nämä tahot tekevät sovelluksen kanssa ja millaisia suhteita heillä on keskenään. On hyvä huomata, että kuvassa on mukana myös sovelluksen ylläpitotoimenpiteet.



Kuva 8. OptiAnalyserin sidosryhmät

3.1.7 Käyttötapaukset

Kuvassa 9 esitetään OptiAnalyserin käyttötapaukset. Kuvassa nähdään päätoimijat eli varastopäällikkö ja muut käyttäjät. On hyvä huomata, että muilla käyttäjillä ei ole oikeutta katsoa varastotyöntekijöiden keräyshistoriaa. OptiAnalyserin käyttäjille voidaan antaa erilaisia oikeuksia, joilla voidaan rajoittaa OptiAnalyserin käyttöä.



Kuva 9. OptiAnalyserin käyttötapaukset

3.1.8 Käyttötapausten läpikäynti

Tässä luvussa käydään läpi käyttötapaukset yleisemmällä selittävällä tasolla, sillä käyttötapausten tarkemmat yksityiskohdat on selitetty liitteessä 1.

Työntekijöiden keräyshistoriadiagrammin ja varastotehtävän tilannediagrammin luonti

Valtuutettu käyttäjä luo diagrammit valitsemalla diagrammivalikosta luontinappulan, jonka jälkeen näytölle ilmestyy dialoginäkymä. Käyttäjä valitsee parametrit dialoginäkymältä ja painaa OK-nappulaa. Kun annetuilla parametreilla löytyy data tietokannassa, niin näkymälle ilmestyy diagrammi. Käyttötapauksen toteutus hyväksytään, jos käyttäjä onnistuu luomaan diagrammin valitsemilla parametreilla.

Diagramminäkymän sijainnin muuttaminen

Kun käyttäjä on luonut diagrammin, niin sen sijaintia pystytään muuttamaan. Sijainti muutetaan siirtämällä hiirtä diagramminäkymän yläpuolelle, jolloin hiiren osoittama nuolikuvake muuttuu siirtymiskuvakkeeksi. Tämän jälkeen käyttäjä klikkaa ja siirtää diagramminäkymä pitämällä klikkausta. Kun klikkaus vapautetaan, näkymä pysyy uudessa paikassa. Käyttötapauksen toteutus hyväksytään, jos sijainnin siirto onnistuu.

Diagramminäkymän koon muokkaus

Kun käyttäjä on luonut diagrammin, niin sen näkymän kokoa pystytään muokkaamaan. Koko muutetaan siirtämällä hiirtä diagramminäkymän oikeaan alanurkkaan, jolloin hiiren osoittama nuolikuvake muuttuu kaksoisnuolikuvakkeeksi. Tämän jälkeen käyttäjä klikkaa ja siirtää hiirensä pitämällä klikkausta, jolloin näkymän koko muuttuu hiiren siirtämissuunnan mukaan. Kun klikkaus vapautetaan, näkymän uusi koko säilyy. Käyttötapauksen toteutus hyväksytään, jos näkymän koon muokkaaminen onnistuu.

Diagrammin poistaminen

Luodut diagrammit voidaan poistaa klikkaamalla dialoginäkymän oikealla yläkulmalla olevaa poistamisnappulaa. Käyttötapauksen toteutus hyväksytään, jos dialogin poistaminen näytöltä onnistuu.

Diagrammin sisällön muokkaaminen

Luotujen diagrammien sisältöä voidaan muokkaa klikkaamalla diagramminäkymässä oleva muokkausnappulaa, jolloin näytölle ilmestyy diagrammin dialoginäkymä, jossa on valittu valmiiksi diagrammin luontivaiheessa valitut parametrit. Käyttäjä muokkaa parametrit ja painaa dialoginäkymän OK-nappulaa, jolloin dialogi- ja vanha diagramminäkymä poistuvat ja tilalle latautuu päivitetty diagramminäkymä. Käyttötapauksen toteutus hyväksytään jos käyttäjä onnistuu muokkaamaan diagrammin sisältöä uusilla parametreilla.

Dashboardin tallentaminen

Luodut diagrammit muodostavat dashbordin, joka voidaan tallentaa selaimen muistiin. Tallennus tapahtuu valitsemalla asetukset-valikosta Tallenna dashboard. Käyttötapauksen toteutus hyväksytään, jos käyttäjä onnistuu tallentamaan dashboardinsa selaimen muistiin.

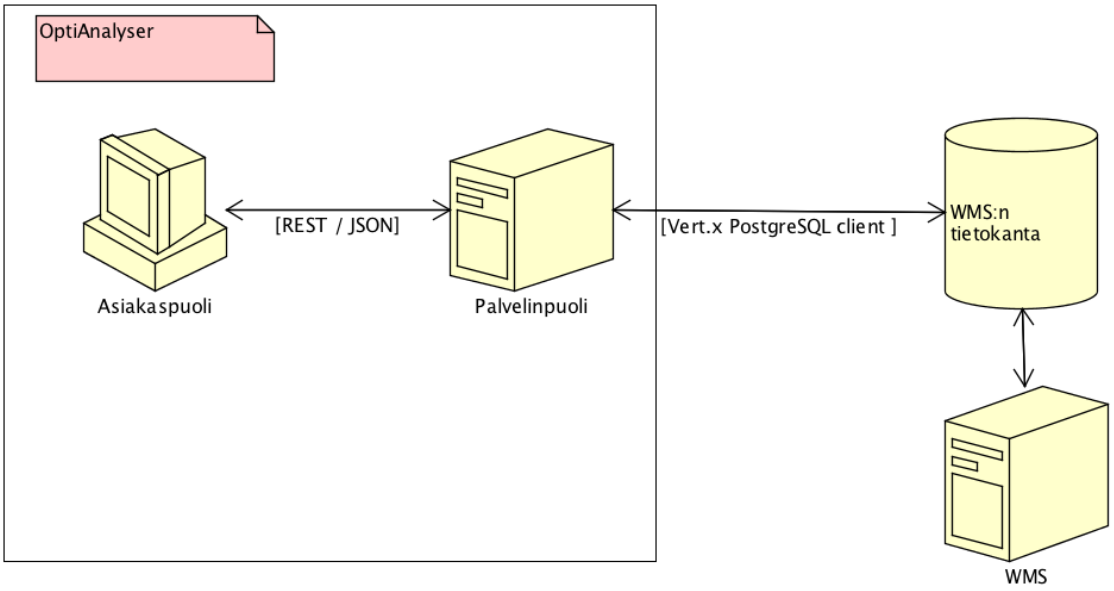
Dashboardin poistaminen

Tallennettu dashboardi voidaan poistaa asetukset-valikon Poista dashboard valinnalla. Poistaminen tapahtuu, kun käyttäjä valitsee tämän, jolloin dashboardi poistetaan selaimen muistista. Käyttötapauksen toteutus hyväksytään, jos käyttäjä onnistuu poistamaan dashboardinsa selaimen muistista.

3.2 Suunnittelu

3.2.1 Sovelluksen järjestelmäarkkitehtuuri

Kuvassa 10 on esitetty OptiAnalyserin järjestelmäarkkitehtuuri. Kuvasta selviää, että OptiAnalyser koostuu kahdesta osasta eli asiakas- ja palvelinpuolesta, jotka keskustelevat toistensa kanssa HTTP/REST-rajapinnan avulla. Asiakaspuolta käyttävät selaimella OptiAnalyserin käyttäjät. Palvelinpuoli tarjoa asiakaspuolelle REST-rajapinnan ja käsittelee sieltä tulevat pyynnöt. Asiakaspuolelta käyttäjän pyynnöt lähetetään REST-kutsuina kuten GET-kutsulla palvelinpuolelle. Palvelinpuoli käsittelee pyynnöt ja hakee tarvittavat tiedot WMS:n tietokannasta. Palvelinpuoli ottaa yhteyttä WMS:n tietokantaan Vert.x:n PostgresSQLclientin avulla.



Kuva 10. OptiAnalyserin järjestelmäarkkitehtuuri

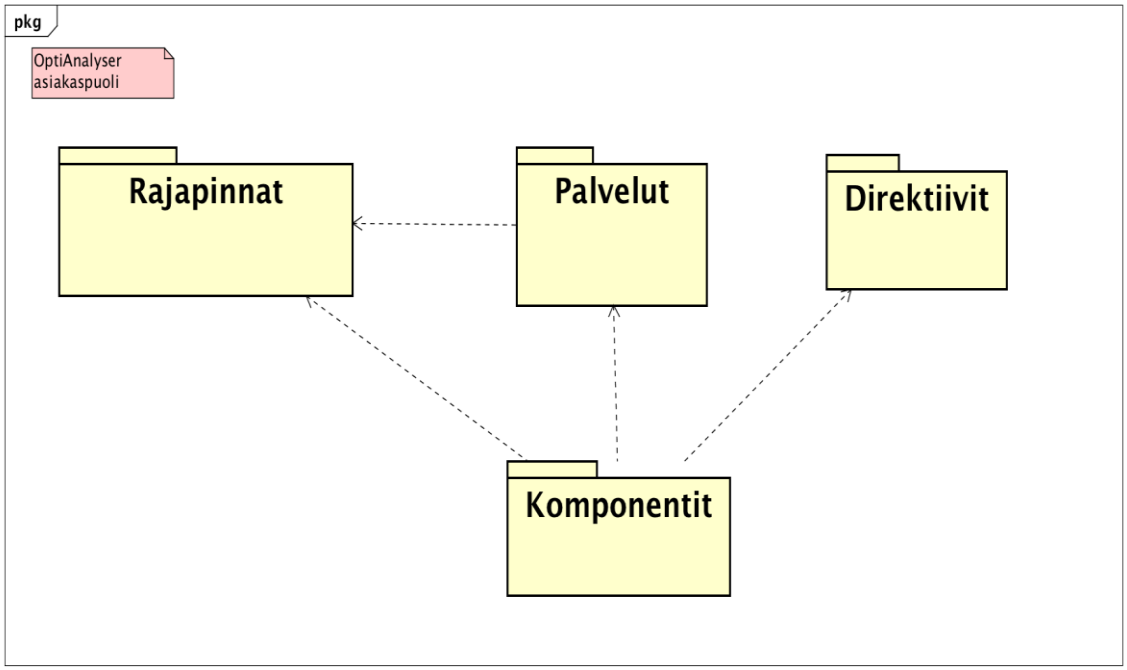
OptiAnalyserissa asiakaspuoli ja palvelinpuoli toteutetaan erillisinä osina. Tämä tarkoittaa sitä, että ne ovat täysin tietämättömiä toisistaan. Tämän lähestymistavan etuna on se, että molemmat puolet voidaan ylläpitää erikseen. Niitä voidaan testata ja korjata erillään toisistaan, niitä voidaan asentaa erillisille palvelimille tai tarvittaessa voidaan helposti korvata toisella toteutuksella. Näin sovelluksen ylläpitokustannukset vähenevät ja mahdolliset jatkokehitykset helpottuvat.

3.2.2 Asiakaspuoli

Pakkauskaavio

Kuvassa 11 nähdään, että asiakaspuoli koostuu rajapinnoista, palveluista, komponenteista ja direktiiveistä. Komponentit käyttävät palveluita, rajapintoja ja direktiivejä. Palveluiden osalta Angular huolehtii niiden injektoinnista komponentteihin. Direktiivejä hyödynnetään injektoimalla niitä komponentin HTML-elementteihin. Näin voidaan muuttaa minkä tahansa elementin käyttäytymistä.

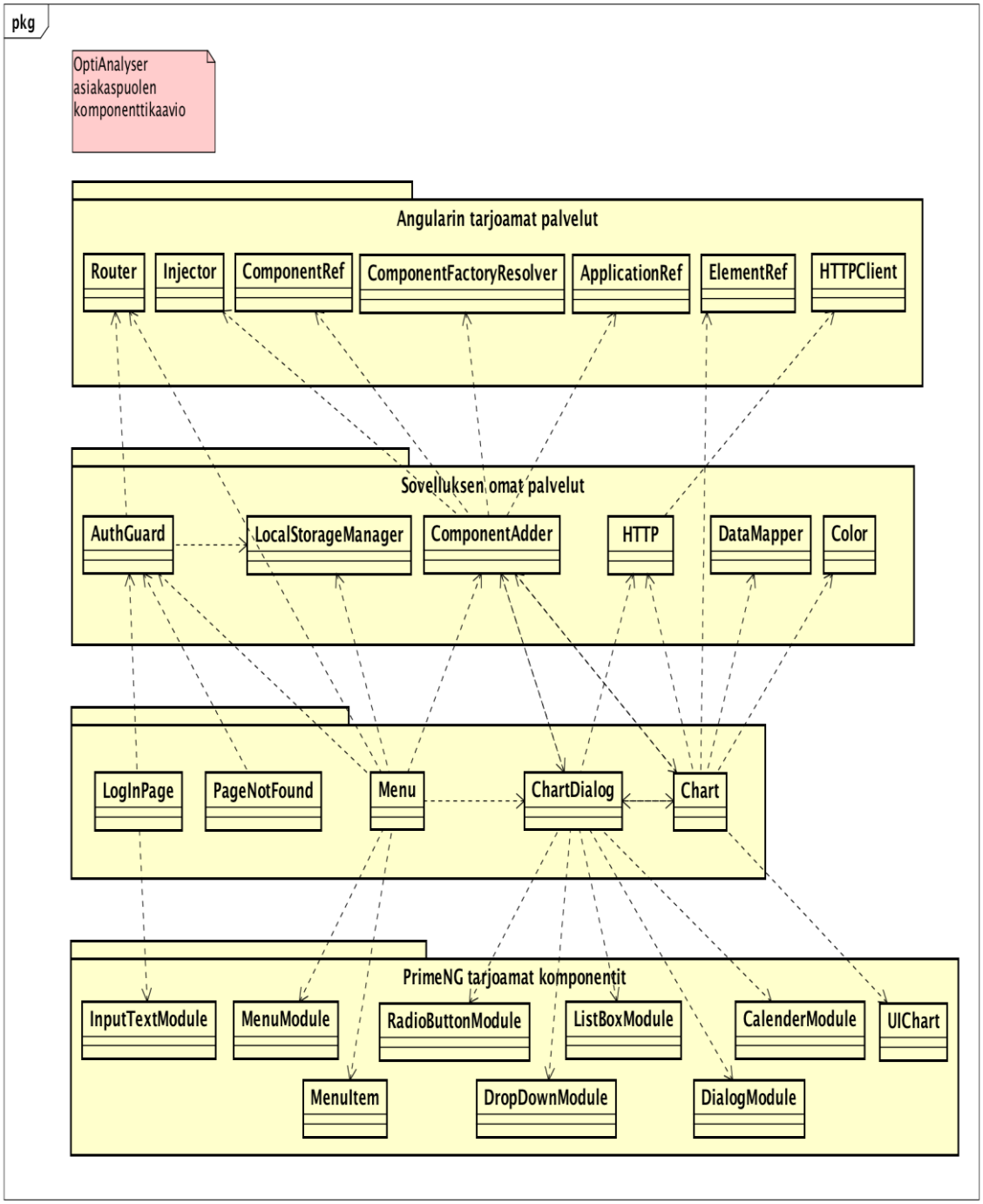
Palvelimelle lähetetään pyynnöt JSON-muodossa ja myös palvelimelta tulevat vastaukset ovat JSON-muodossa. Kuitenkin JSONia ei kannattaa käsitellä sellaisena toistuvasti. Rajapintapakkaus sisältää rajapintoja, joiden avulla voidaan palvelimelta saatu JSON-muodossa oleva vastaus kuvata olioksi, sillä tällä tavalla sovelluksessa vastauksen käsittely helpottuu merkittäväksi. OptiAnalyserissa sekä komponentit että palvelut käyttävät näitä rajapintoja.



Kuva 11. OptiAnalyserin asiakaspuolen pakkauskaavio

Komponenttikaavio

Kuvassa 12 nähdään asiakaspuolen komponenttien ja palveluiden väliset riippuvuudet. On hyvä huomata, että jotkut komponentit käyttävät suoraan Angularin tarjoamia palveluita.



Kuva 12. OptiAnalyserin asiakaspuolen komponenttikaavio

Komponentit

Asiakaspuoli koostuu viidestä komponentista. Nämä komponentit ovat Chart, ChartDialog, Menu, PageNotFound ja LogIn.

Chart-komponentin avulla luodaan erilaisia diagrammeja. Komponentti sisältää diagramminäkymän HTML-koodin, diagramminattribuutit sekä muokkaa ja poista -nappulat. Komponentti käyttää HTTP-palvelua REST-pyyntöjen lähettämiseksi palvelimelle. Rajapintojen avulla JSON-muodossa olevat vastaukset muutetaan oliopohjaisiksi. DataMapper-palvelun avulla kuvataan palvelimelta saatu data komponentin kentille ja Color-palvelun avulla komponentti saa diagrammiinsa värejä. Angularin ElemntRef-palvelun avulla saadaan komponentin metadataa. Chart- ja ChartDialogkomponenti luodaan dynaamisesti ComponentAdder-palvelulla.

ChartDialogkomponentin avulla kerrotaan ComponentAdder-palvelulle, millainen diagrammi halutaan luoda. Komponentti sisältää dialoginäkymän HTML-koodin, parametrien kentät ja nappulat. Komponentti hakee satavilla olevat parametrit HTTP-palvelun avulla palvelimelta. Näiden parametrien avulla käyttäjä luo haluamansa diagrammin.

Menu komponentin avulla rakennetaan sovelluksen valikko. Komponentti sisältää valikon HTML-koodin, nappulat ja niiden toiminnallisuuslogiikan. Käyttäjän valinnan perusteella Menu kertoo ComponentAdder-palvelulle, minkä tyyppisen ChartDialog-komponentin sen tulisi luoda. LocalStorageManager-palvelun avulla komponentti tallentaa käyttäjän dashboardin selaimen muistiin sekä pyydettäessä poistaa sen sieltä. Valikossa on myös uloskirjautuminen toiminto. Uloskirjautuessa valikko ohjaa käyttäjän sisäänkirjautumissivulle Angularin reititys (Router) palvelun avulla.

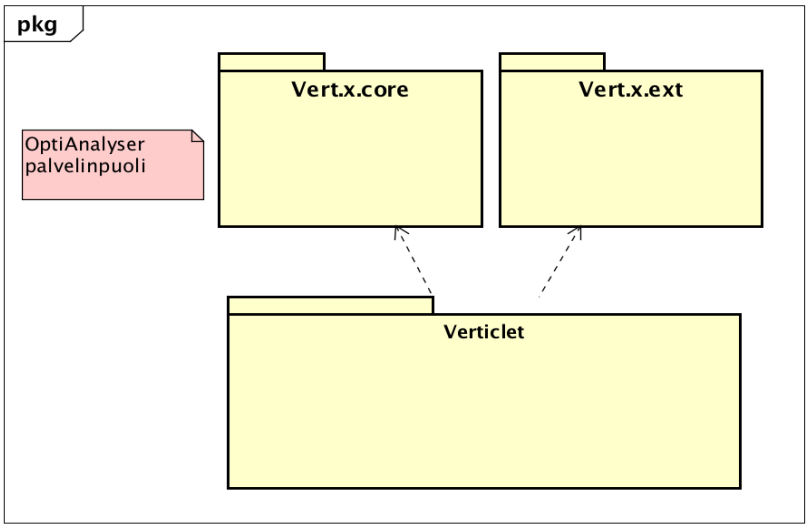
LogIn-komponentin avulla rakennetaan sisäänkirjautumissivua. Komponentti sisältää sivun HTML-koodin, tekstikentät ja sisäänkirjautumisen logiikan. Komponentti validoi syötetyt tunnukset ja estää asiattomien pääsyä sovellukselle AuthGuard-palvelun avulla, jossa tarkistetaan, onko käyttäjällä oikeutta kirjautua. Kun käyttäjä pääsee validoinnista läpi, komponentti ohjaa hänet DashboardSivulle, jossa sijaitsee Menu-komponentti.

PageNotFound-komponentin avulla rakennetaan virhesivu, joka käytetään silloin, kun käyttäjä kirjoittaa sovelluksen domainia sisältävän virheellisen URL:n.

3.2.3 Palvelinpuoli

Pakkauskaavio

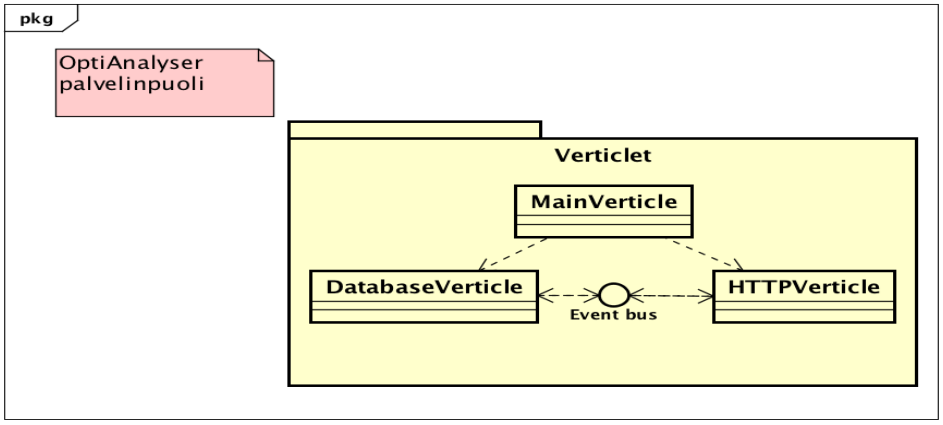
Kuvassa 13 nähdään, että palvelinpuolen pakkauskaavio on hyvin yksinkertainen. Se koostuu verticle-pakkauksesta ja Vert.x:n tarjoamista pakkauksista. Verticle-pakkaus käyttää Vert.x:n tarjoamia palveluita, joita ovat Vert.x.core ja Vert.x.ext.



Kuva 13. OptiAnalyserin palvelinpuolen pakkauskaavio

Komponenttikaavio

Kuvassa 14 nähdään, että OptiAnalyserin palvelinpuoli koostuu kolmesta komponentista eli verticleistä. Verticlet keskustelevat toiselle yhteisen osoitteen ja event busin avulla.



Kuva 14. OptiAnalyserin palvelinpuolen komponenttikaavio.

Palvelinpuoli toteutetaan kolmella verticlellä. Nämä ovat MainVerticle, DatabaseVerticle ja HTTPVerticle. MainVerticlen tehtävänä on hallinnoida DatabseVerticlen ja HTTPVerticlen luontiprosessia. Verticleiden luontia varten MainVerticle käyttää Vert.x-ytimen tarjoamia palveluita.

HTTPVerticlen tehtävänä on luoda REST-rajapinta ja siihen liittyvät reitit, joiden avulla asiakaspuolen kanssa keskustellaan. HTTP-palvelin luodaan käyttäen Vert.x:n tarjoamia palveluilita. HTTPVerticle koostuu käsittelijöistä, jotka käsittelevät asiakaspuolelta tulevia REST-pyyntöjä. HTTPVerticle tarvittaessa pyytää EventBusin avulla DatabaseVerticleä suorittamaan tietokantakyselyitä ja välittää sen jälkeen tulokset asiakaspuolelle JSON-muodossa.

DatabaseVerticlen tehtävänä on luoda WMS:n tietokantayhteyttä varten tietokannan asiakaskomponentti (databse client). Tietokanta-asiakas luodaan Vert.x:n PostgresSqlClientin avulla. DatabseVerticle käsittelee HTTPVerticlestä tulevia tietokantakyselypyyntöjä ja suorittaa niitä. Kyselyiden tulokset välitetään JSON-muodossa HTTPVerticlelle Event busin avulla.

3.3 Toteutus

3.3.1 Asiakaspuolen toteutus

Tässä luvussa käydään läpi asiakaspuolen toteutuksen vaiheita. Käydään läpi komponenttien rakennetta sekä tärkeimpien palveluiden sisältöä.

3.3.2 Angularin asennus ja projektin luonti

Kehitystöiden helpottamiseksi käyttöjärjestelmälle asennettiin Angularin CLI-työkalu, joka on komentoriviliittymä Angularille. Työkalun avulla voidaan luoda komponentteja ja palveluita. Angular CLI:n avulla sovellukselle luodaan oma palvelin, johon otetaan selaimella yhteyttä. Jokaisen tallennetun koodimuutoksen jälkeen Angular CLI päivittää muutokset automattisesti sovelluksen palvelimelle, joten muutokset ovat heti nähtävissä selaimella eikä erillisiä toimenpiteitä tarvita. Angular CLI asennetaan npm-paketinhallintatyökalulla (esimerkkikoodi 9). Angular 2 ja sitä uudemmat versioit tarvitsevat toimiakseen Node.js:ää, joka on JavaScript ajonaikainen ympäristö. Node.js

tarvitaan, koska Angular 2 ja uudemmat versiot kirjoitetaan TypeScript-ohjelmointikielellä, mutta selaimet eivät ymmärrä sitä, vaan ne ymmärtävät ja tukevat JavaScriptiä. Node.js:n avulla TypeScript-koodi käännetään sellaiseen muotoon, jota selain ymmärtää.

```
npm install -g @angular/cli
```

Esimerkkikoodi 9. Angular CLI -työkalun asennus npm:llä Linuxin komentoriviltä.

Asennuksen jälkeen voidaan generoida projektipohja, jota käytetään OptiAnalyserin toteuttamiseen. Angular CLI luo automattisesti kaikki tarvittavat tiedostot (esimerkkikoodi 10).

```
ng new OptiAnalyser
```

Esimerkkikoodi 10. Projektipohjan generointi Angular CLI-työkalulla

Tuloksena saadaan projektipohja, jossa on seuraavalainen tiedostorakenne:

Kuva 15. Angular-projektin tiedostorakenne

Valikkokomponentti

Asiakaspuolen toteutus aloitettiin sovelluksen valikosta. Angular CLI:n avulla luotiin valikkokomponentti. Se generoi automaattisesti tarvittavat tiedostot komponentille ja rekisteröi sen app-moduuliin, jotta se olisi saatavilla moduulin muille komponenteille (Esimerkkikoodi 11). Angular CLI sijoittaa generoidut komponentit projektin /src/app-hakemistoon. App-moduuli on sovelluksen päämoduuli, joka sisältää kaikki palvelut ja komponentit.

```
// komponentin generointi
ng generate component Menu
  create src/app/menu/menu.component.html (23 bytes)
  create src/app/menu/menu.component.spec.ts (614 bytes)
  create src/app/menu/menu.component.ts (261 bytes)
  create src/app/menu/menu.component.css (0 bytes)
```

Esimerkkikoodi 11. Komponentin generointi Angular CLI -työkalun avulla. Menu.component.spec.ts-tiedostoa käytetään komponentin testaamiseen.

Esimerkkikoodissa 12 nähdään komponentin logiikkaluokka, missä ennen luokan määrittelyä on määritelty @Component-kuorruttaja (decorator). Tämän avulla komponentin muut tiedostot kytketään yhteen ja komponentille annetaan valitsija (selector).

```
@Component({
  selector: 'app-menu', // tämän avulla voidaan upottaa komponenttia muihin
                        // koponentteihin.
  templateUrl: './menu.component.html', // Tämän komponentin HTML-tiedoston
                                        // sijainti.
  styleUrls: ['./menu.component.css'],  // Tämän komponentin CSS-tiedoston
                                        // sijainti.
})
export class MenuComponent{}
```

Esimerkkikoodi 12. Komponentinmäärittely Angularissa.

Komponentin generoinnin jälkeen valikkoa varten etsittiin sopiva UI-komponentti Angularin kanssa yhteensopivista UI-komponenttikirjastoista. Ratkaisuksi päädyttiin käyttämään PrimeFacesin tarjoamaa PrimeNg:n UI-kirjastoa. PrimeNg:llä on laaja komponenttitarjonta Angularille, ja se perustuu avoimeen lähdekoodiin. [17.]

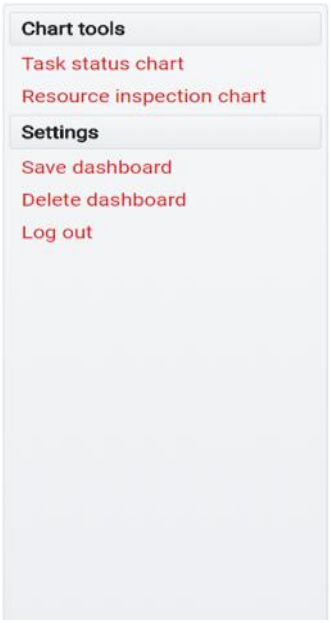
PrimeNg-kirjasto tarjoaa Menu-komponentin, jonka avulla rakennetaan erilaisia valikoita. Komponenttia käytetään rekisteröimällä se app-moduuliin. Tämän jälkeen voidaan käyttää komponentin HTML-elementtiä OptiAnalyserin valikkokomponentin HTML-mallipohjassa (esimerkkikoodi 13).

```
menu.component.html
<p-menu [model]="items"></p-menu>
```

Esimerkkikoodi 13. PrimeNg:n menukomponentin lisääminen valikkokomponentin HTML-tiedostoon.

Menukomponentti käyttää kokoelmaa valikkokohteista. Valikkokohde on tyypiltään MenuItem-tyyppi, joka on toteutettu PrimeNG:ssä. Kokoelma määritellään valikkokomponentin logiikkaluokassa.

OptiAnalyserin valikossa on kaksi päävalikkokohdetta, joilla on oma alavalikko. Nämä ovat diagrammityökalu (chart tools) ja asetukset (settings). Jokaiselle valikkokohteelle määritellään alivalikko, jolloin saadaan pääkohteiden ominaisuudet niiden alle. Diagrammityökalun alle sijoitetaan dialogikomponentin luontinappulat. Puolestaan asetusten alle laitetaan sovelluksen muut toiminnallisuudet, kuten dashboardin tallentaminen ja poistaminen sekä uloskirjautuminen. Alivalikon nappulat kytketään luokan metodeihin valikkokohteiden command-metodin avulla.

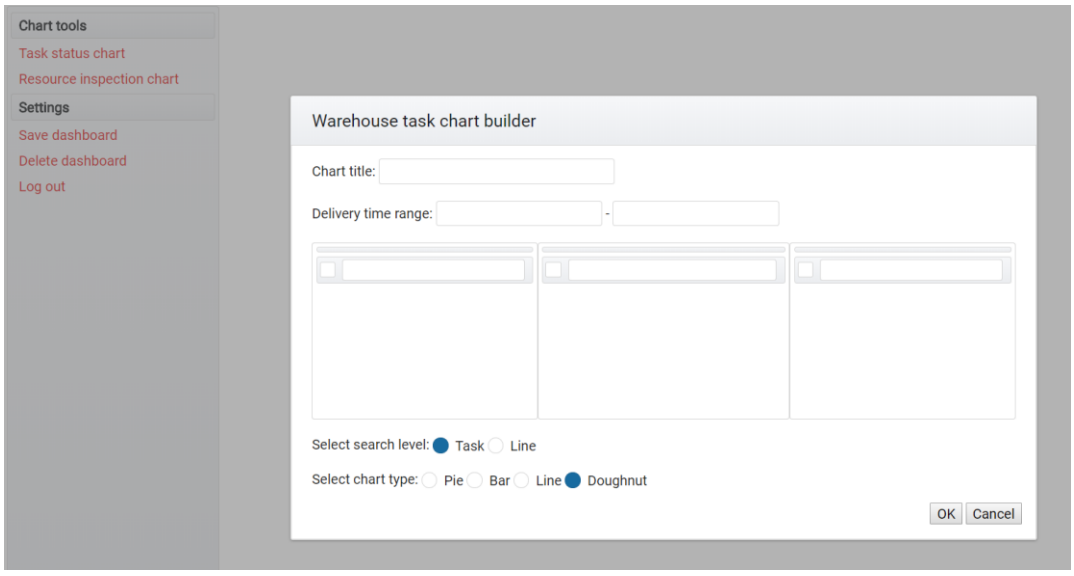


Kuva 16. OptiAnlyserin valikko.

Dialogikomponentti ja ComponentAdder-palvelu

Dialogikomponentin tehtävänä on tarjota käyttäjälle parametreja diagrammin luontia varten. Dialogikomponentin generoinnin jälkeen etsittiin sopiva UI-komponentti PrimeNg:stä. Sieltä löydettiin dialogikomponentti, joka oletuksena sisälsi tekstikentän sekä kyllä- ja ei-nappulat. Tätä käytettiin pohjana ja siihen lisättiin tarvittavat lisäelementit PrimeNg-UI -kirjastolta.

OptiAnalyserin dialogikomponentti koostuu yhdestä tekstin syöttökentästä, kahdesta kalenterimoduulista, kolmesta listbox-moduulista, kuudesta radiobutton-modulista sekä OK- ja Peru (cancel) -nappulasta. Käyttäjä syöttää tekstin syöttökenttään diagrammin nimen. Kalenterimoduulien avulla rajataan hakutulokset tietyn tilauksen toimitusaikaväliin. Listbox-moduuli näyttää käyttäjälle saatavilla olevat parametrit kuten varastoalueet, tilaustyypit ja tilauksen tilan. Näitä parametreja haetaan palvelimelta. Radiobutton-valintojen avulla käyttäjä määrittää haun tason sekä diagrammiin ulkonäkötyypin. Dialoginäkymä näyttää täältä:



Kuva 17. Dialogikomponentin näkymä.

OptiAnalyserin tässä versiossa voidaan luoda dialoginkomponentin avulla kahta eri tyyppistä diagrammia, jotka ovat varastotehtävien tilannediagrammit ja varastokerääjien keräyshistoriaa kuvaava resurssien tarkasteludiagrammi. Kuvassa 17 nähdään varastotehtävien tilannediagrammia varten luotu dialoginäkymä ja sen kentät. Vaikka diagrammit luodaan erilaisilla parametreilla ja eri sisältöisellä dialoginäkymällä, kuitenkin

tehokkaamman toteutuksen kannalta päätettiin käyttää samaa dialogikomponenttia molemmille diagrammityypeille. Tämä saavutettiin muuttamalla dialogikomponentin sisältöä dynaamisesti niin, että jokaiselle dialogikomponentin luontivaihtoehdolle annettiin oma tyyppi. Luontityyppi määräytyy sen mukaan kumman diagrammin luontinappulaa käyttäjä painaa valikolta. Dialogikomponentissa katsotaan valintalauseella luontipyynnön tyyppi ja sen mukaan sisältöä muokataan.

Dialogikomponentti luodaan, kun käyttäjä valitsee diagrammityökalun valikolta jompaakumpaa diagramminluontianappulaa. Menukomponentti välittää luontipyynnön ja dialogityypin ComponentAdder-palvelulle, joka luo dialogikomponentin dynaamisesti käyttäen Angularin tarjoamia palveluita kuten ApplicationRef, ComponentFactoryResolver, componentRef. ComponentFactoryResolverin avulla luodaan komponentin instanssi. componentRefin avulla säilytetään luodun komponentin referenssi. ApplicationRefin avulla injektoidaan komponentti Angularin komponenttipuuhun. ComponentAdder-palvelu antaa komponentille tunnisteen ja säilyttää niitä komponenttien tyyppien mukaisesti hakurakenteissa. Näin komponentteja voidaan tarvittaessa helposti muokata. Näin sovelluksen tehokkuus kasvaa, sitä ei tarvitsisi poistaa ja luoda komponentteja uudestaan.

Diagrammikomponentti

Diagrammikomponentin (ChartComponent) avulla luodaan diagrammeja. Komponentin generoinnin jälkeen etsittiin PrimeNg:stä sopiva diagrammikomponenttia. PrimeNG tarjoaa ChartModuulin, joka perustuu Chart.js 2.7.x -kirjastoon. Chart.js perustuu avoimeen lähdekoodiin ja HTML5:een. [18.]

Jotta voittaisin käyttää Chart.js projektissa, tulisi se asentaa ensin projektille. Asennus tehdään npm:n avulla. Asennuksen jälkeen Chart.js:n skripti tulisi lisätä projektin angular-cli.json -tiedoston scripts-taulukkoon. Niin kuin kaikki muutkin komponentit, jotta ChartModuulia voittaisin käyttää projektissa, se tulisi rekisteröidä projektin app.module.ts-tiedoston import-taulukkoon. Näiden vaiheiden jälkeen voidaan käyttää ChartModuulia diagrammikomponentissa. ChartModuulia käytetään lisäämällä se elementiksi diagrammikomponentin HTML-tiedostoon (esimerkkikoodi 14).

```
<p-chart #chart type={{chartType}} name="chart" [data]="data" [options]="op-
tions"</p-chart>
```

Esimerkkikoodi 14. p-chart elementin lisääminen chart.component.html-tiedostoon. Tässä nähdään myös, miten Angularissa HTML-attribuutit sidotaan luokkamuuttujiin. Esimerkkinä `[data]="data"` tarkoittaa sitä, että data-attribuutti saa arvonsa chart.component.ts-tiedostossa olevalta data-nimiseltä luokkamuuttujalta. Puolestaan Angularissa tupla-aaltosulut käytetään silloin, kun halutaan sijoittaa logiikkaluokassa olevan merkkijonomuuttujan arvo HTML-attribuutin arvoksi.

Diagrammikomponentti luodaan sen jälkeen, kun käyttäjä on valinnut diagramminäkymältä parametrit. Dialogikomponentti välittää luontipyynnön ComponentAdder-palvelulle, joka luo diagrammikomponentin dynaamisesti. Luonnin jälkeen ComponentAdder-palvelu asettaa diagrammille dialoginäkymältä valitut parametrit ja säilyttää sitä hakurakenteessa. Diagrammikomponentissa valitut parametrit käsitellään ja tarvittava data pyydetään palvelinpuolelta HTTP-palvelun avulla. Diagrammin käyttämä data koostuu etiketistä ja etiketin arvosta. Esimerkiksi pylväsdiagrammin tapauksessa arvo olisi y-koordinaatti ja etiketti x-koordinaatti.

HTTP-palvelu

HTTP-palvelu käyttää Angularin tarjoamaa HTTPClienttia. Tämän avulla voidaan suorittaa REST-pyyntöjä ja saada palvelimelta dataa. HTTP-palvelu tarvitsee myös muita Angularin tarjoamia palveluita kuten HttpHeaders ja HttpParams pyynnön tekemiseen. HttpHeadersin avulla pyynnölle liitetään otsikoita (headers), jotka kertovat palvelimen vaativia informaatiota. HttpParamsin avulla pyynnölle lisätään parametreja kuten esimerkiksi käyttäjän id tai ikä.

Palvelu generoitiin Angular CLI:n avulla:

```
// palvelun generointi
ng generate service http
create src/app/http.service.spec.ts (362 bytes)
create src/app/http.service.ts (110 bytes)
```

Esimerkkikoodi 15. Palvelun generointi Angularissa.

Angular luo automaattisesti palvelun luokan ja lisää @Injectable-kuorruttajan ennen luokan määrittelyä (esimerkkikoodi 15). Tämä kertoo Angularille, että tämä luokka on injektoitavissa komponentteihin ja muihin palveluihin. Käytännössä Angular luo palvelusta yhden instanssin (singleton). Komponentit käyttävät palveluita niin, että komponentin konstruktoriin lisätään haluttu palvelu parametrina (esimerkkikoodi 16).

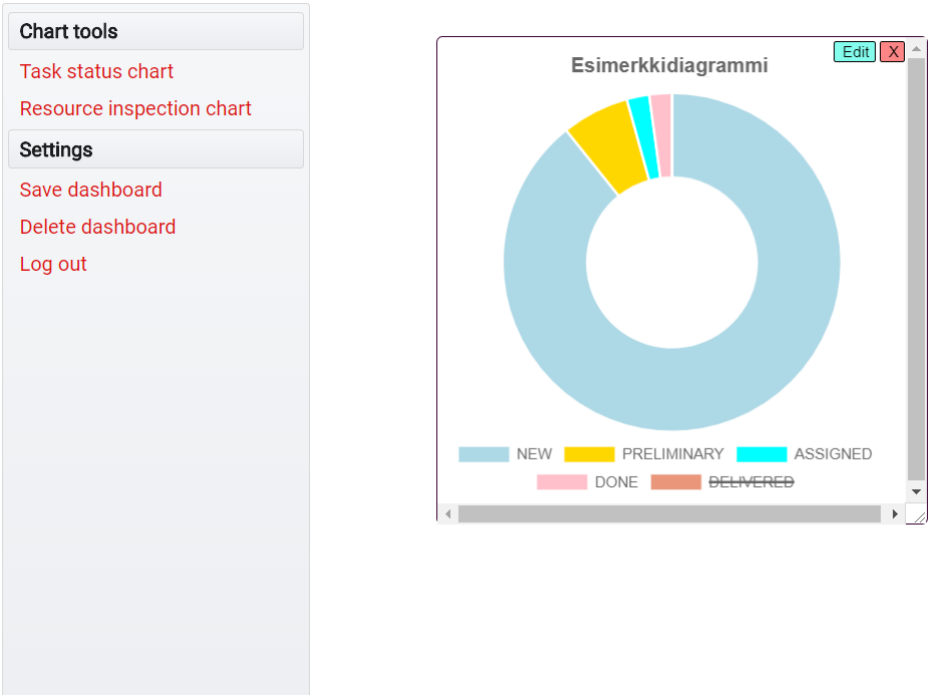
```
import { HttpService } from '../../services/http-service/http.service';
class ExampleComponent {
    constructor(private httpService: HttpService);
   }
}
```

Esimerkkikoodi 16. Palvelun injektointi komponenttiin. Tässä on hyvä huomata, että TypeScriptillä ei tarvita hae- ja aseta-metodeja (getter and setter) konstruktorin yksityisiin parametreihin.

Kun diagrammikomponentti saa palvelimelta vastauksen, se kuvaa sen etiketiksi ja arvoksi diagrammin esitystä varten. Tässä tapauksessa etiketti on varastotehtävän statuksen nimi, ja arvo on varastotehtävien määrä. Arvo esitetään diagrammintyypin mukaan joko numeroina tai pinta-alana (kuva 18). Tämä kuvaus tapahtuu DataMapper-palvelun avulla, jossa WarehouseTaskStatuses-olioksi muutettu data puretaan. DataMapper-luokassa on metodi mapWarehouseTaskStatusOrResourceData, joka tekee kuvauksen (esimerkkikoodi 17).

```
/**
  * Metodi kuvaa WarehouseTaskStatuseksi muutettu dataa labeliksi ja arvoksi.
  * @param data. data, joka kuvataan.
  */
  mapWarehouseTaskStatusOrResourceData(data: WarehouseTaskStatuses):
  Map<string, any[]> {
    const dataMap = new Map<string, any[]>();
    const labels = new Array<string>();
    const values = new Array<number>();
    for (let i = 0; i < data.statuses.length; i++) {
      labels[i] = data.statuses[i].statusName;
      values[i] = data.statuses[i].statusCount;
    }
    dataMap.set('labels', labels);
    dataMap.set('values', values);
    return dataMap; }
```

Esimerkkikoodi 17. Datan kuvaaminen labeliksi ja arvoksi DataMapper-palvelun avulla.



Kuva 18. Diagramminäkymä.

Diagrammikomponentin siirtäminen ja koon muuttaminen

Käyttäjä rakentaa dashboardinsa siirtämällä ja muokkaamalla diagrammien kokoa. Diagrammien koon muuttaminen toteutetaan käyttämällä CSS:n muokkausominaisuutta. Diagrammin elementti sijoitetaan div-elementin sisään, jonka mittoja voidaan muuttaa asettamalla sille resize-attribuutti. Kuitenkin diagramminäkymän siirtämiselle ei ollut CSS-ratkaisua. Siihen keksittiin ratkaisu käyttämällä direktiiviä. Angularissa direktiivien avulla voidaan muuttaa DOM-elementin käyttäytymistä. Tässä tapauksessa tarvittiin sellainen direktiivi, jonka avulla voitaisiin rekisteröidä hiiren koordinaatisto ja käyttää sitä diagrammin sijainnin muuttamista varten.

Direktiiviluokka toteutettiin niin, että se rekisteröi käyttäjän klikkauksen diagrammin päällä, ja kun käyttäjä siirtää hiirtä pitämällä klikkausta, niin diagrammin sijainti muuttuu sen mukaan. Tämä onnistuu @HostListener-tapahtumakuuntelijan avulla, johon välitetään hiiren klikkaustapahtuma (event) ja klikatun elementin referenssi (elem). Siirtologiikka toimii niin, että kun käyttäjä siirtää hiirensä ja vapauttaa klikkauksen, niin direktiivi laskee siirron määrää ja sen mukaan päivittää näkymän sijaintia. Direktiiviä käytetään niin, että diagrammikomponentin HTML-tiedostoon lisätään näkymätön div-elementti, koska siirto sallitaan vain, jos käyttäjä siirtää hiirensä tämän divin päälle.

Direktiivi injektoidaan tähän div-elementtiin, joka puolestaan sijaitsee toisessa div-elementissä, joka sisältää diagrammielementin (esimerkkikoodi 18).

```
<div id = "diagramDiv">
    <div draggable id="draggableDiv"></div>
    <button (click)="editChart()">Edit</button>
    <button (click)="closeChart()">X</button>
    <p-chart #chart type={{chartType}} name="chart" [data]="data"
    [options]="options"></p-chart>
</div>
```

Esimerkkikoodi 18. Draggable-direktiivin injektointi div-elementiin.

Diagrammikomponentin muokkaus ja poisto

Diagrammikomponentissa on kaksi nappulaa, jotka ovat muokkaus (Edit) ja poisto (X). Poistonappulan avulla diagrammi poistetaan dynaamisesti Angularin komponenttipuusta. Poistaminen tapahtuu ComponentAdder-palvelun avulla.

Diagrammikomponentin poistonappula kutsuu diagrammikomponentin logiikkaluokassa olevaa CloseChart-metodia. Tämä metodi pyytää ComponentAdder-palvelusta diagrammikomponentin poistamista välittämällä parametrina komponentin id:n. ComponentAdder-palvelussa diagrammikomponentti poistetaan hakemalla sen referenssiä id:n perusteella hakurakenteesta, johon komponentin referenssi lisättiin luonnin yhteydessä. Komponentti poistetaan ensin hakurakenteesta ja sen jälkeen Angularin komponenttipuusta applicationRef-palvelun avulla.

Muokkausnappulan avulla muokataan diagrammin sisältöä uusilla parametreilla. Kun käyttäjä painaa muokkausnappulaa, niin avautuu dialoginäkymä, joka käytettiin diagrammin luontia varten. Ero on siinä, että nyt käyttäjän valinnat muistetaan ja ne ovat valittuina valmiiksi dialoginäkymässä. Tämä on erityisen hyödyllistä, jos muokattavaa on vähän ja parametreja on valittu suhteellisen paljon.

Muokkaustoiminnallisuus toteutettiin niin, että muokkausnappulaa kutsuu diagrammikomponentin logiikkaluokassa oleva Editchart-metodia ja välittäen sille diagrammin parametrit. Editchart hakee dialogikomponentin ComponentAdder-palvelun avulla ja asettaa sille parametrit. Dialogikomponentille välitetään diagrammikomponentin id, jotta se päivittäisi tiedot oikeaan diagrammiin.

Dashboardin tallentaminen ja poisto

Käyttäjän pitäisi pystyä tallentamaan dashboardinsa muistiin, jotta joka kerta ei tarvitsisi luoda diagrammeja uudestaan. OptiAnalyserilla ei ole oma tietokanta, joten tässä versioissa dashboardi tallennetaan ainoastaan selaimeen muistiin. Tämä toiminnallisuus toteutettiin LocalStorageManager-palvelun avulla. Tämän palvelu tehtävänä on merkkijonopohjaisen datan tallennus selaimen muistiin.

Kun käyttäjä painaa valikolta tallenna dashboard -nappulaa niin luodut diagrammit tallennetaan selaimeen muistiin niin, että diagrammit muutetaan JSON-muotoon, koska selaimeen muistiin pystytään tallentamaan ainoastaan merkkijonoja. Diagrammin sisältöä ei kuitenkaan tallenneta vaan diagramminluontia varten tarvittavat parametrit. Kun dashboardi ladataan, niin parametreista rakennetaan diagrammit uudestaan. Muistiin tallennetaan myös diagrammien koordinaatit, joten diagrammit menevät omille paikoilleen. Jos dashboardia muokataan ja tallennetaan uudestaan, niin tallennuslogiikka poistaa ensin vanhan dashboardin muistista ja tallentaa sen tilalle uuden dashboardin. Tämä johtuu siitä, että muutoksia ei synkata vanhan dashboardin kanssa. Puolestaan kun käyttäjä painaa poista dashboard -nappulaa, niin selaimen muistista poistetaan tallennettu dashboardi kokonaan eikä sitä voi enää palauttaa.

Sisäänkirjautuminen ja reititys

Sisäänkirjautuminen tapahtuu LogIn-komponentin avulla, jossa on kaksi tekstin syöttökenttää tunnuksen ja salasanan syöttämiseen. LogIn-komponentti validoi tunnukset palvelinpuolen avulla WMS-tietokannan kanssa. Palvelinpuoli luo ja lähettää käyttöoikeustietueen asiakaspuolelle, jos tunnukset täsmäävät. Tämä tietue on uniikki jokaiselle käyttäjälle. Sitä käytetään mm. dashboardin tallennuksessa. LogIn-komponentti tallentaa tämän tietueen selaimen muistiin LocalStorageManager-palvelun avulla.

Angularin reitityspalvelun avulla käyttäjä ohjataan dashboard-sivulle kirjautumisen jälkeen. Puolestaan kun käyttäjän pyytämä sivu ei löydy, niin hänet ohjataan PageNotFound-sivulle, jossa käyttäjälle ilmoitetaan virheestä. Dashboard-sivulle pääsyä rajataan AuthGuard-palvelun avulla, joka tarkistaa selaimen muistslta, onko käyttöoikeustietue olemassa vai ei. Jos tietuetta ei löydy, niin käyttäjä ohjataan LogIn-sivulle (esimerkkikoodi 19).

```
const appRoutes: Routes = [
  { path: 'login', component: LogInComponent},
  { path: 'dashboard', canActivate: [AuthGuard],
    component: BasicMenuComponent},
  { path: '**', component: PageNotFoundComponent }
];
```

Esimerkkikoodi 19. Reittien määrittäminen Angularissa. Reitit määritellään app.module.ts-luokassa.

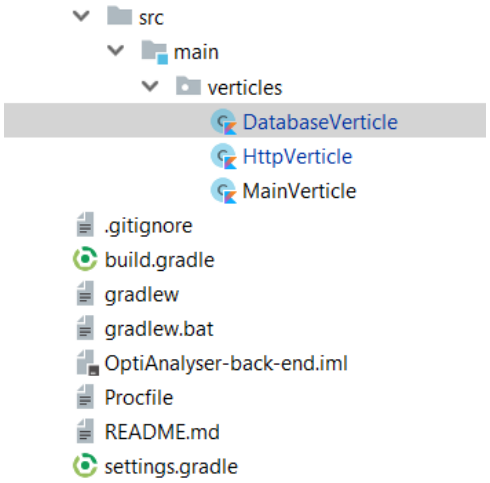
3.3.3 Palvelinpuolen toteutus

Tässä luvussa käydään läpi palvelinpuolen toteutusta. Tarkastellaan mm. miten HTTP-palvelin luodaan Vert.x:n avulla, miten verticlet luodaan ja mitä ne tekevät sekä miten ne kommunikoivat EvenBusin avulla.

3.3.4 Ohjelmointiympäristö ja projektirakenne

Toteutusta varten ohjelmointiympäristöksi valittiin IntelliJ IDEA:n ilmainen versio, sillä se tukee Kotlinia ja sillä on tähän projektiin tarvittavat kaikki ominaisuudet. IntelliJ:n ilmainen versio perustuu avoimeen lähdekoodin ja on saatavilla suoraan IntelliJ:n sivulta. [19.]

IntelliJ:n avulla luotiin Gradle-projekti, johon lisättiin Vert.x:n core-kirjasto riippuvuuksina build.gradle-tiedostoon. Projektiin lisättiin verticles-pakkaus ja sinne luotiin kolme Kotlin-luokkaa. Saatiin kuvan 19 näköinen rakenne.



Kuva 19. Palvelinpuolen projektirakenne

3.3.5 Verticlet

MainVerticle

Palvelinpuoli koostuu kolmesta verticlestä. Nämä ovat MainVerticle, HTTPVerticle ja DatabseVerticle. Vert.x:ssä kaikki verticleiden tulisi periä Vert.x.coren AbstractVerticle-luokka. Vert.x:lle kerrotaan MainVerticlen sijainti build.gradle-tiedostossa, sillä ohjelman suoritus alkaa MainVerticlestä. Tämän verticlen tehtävänä on luoda HTTP- ja DatabaseVerticle niin, että HTTPVerticle luodaan vain, jos DatabaseVerticlen luonti onnistuu. DatabaseVerticlen luonti onnistuu, jos verticle saa yhteyden WMS:n tietokantaan. Verticlet luodaan ja otetaan käyttöön vertx-olion avulla, jonka MainVerticle peri AbstractVerticle-luokasta. Vertx-oliota ei tarvitse itse luoda vaan sen luo automaattisesti Vert.x:n ydin.

HTTPVerticle

HTTPVerticlen tehtävänä on luoda HTTP-palvelin ja käsitellä sieltä tulevat pyynnöt. Palvelin luodaan vertx-olion avulla, jonka HTTPVerticle peri AbstractVerticle-luokasta. Palvelimen luontia varten täytyy ensin määrittää reitit ja käsittelijät. Router-palvelun avulla reitit luodaan. Reitille annetaan parametrina sen käsittelijä. Käsittelijät ovat asynkronisia, ja ne vastaavat palvelinpuolelle vasta, kun ne saavat DatabaseVerticleltä vastauksen.

DatabaseVerticle

DatabseVerticle luo postgresSQLClientin, jonka avulla palvelinpuoli ottaa yhteyttä WMS-tietokantaan. Verticle kuuntelee myös EventBusin kautta tulevia viestejä ja käsittelee ne. Mikä tahansa luokan metodi voi olla kuuntelijametodi. Kun EvenBusin avulla lähetetään viestejä, sille kerrotaan, mikä on Verticlen kuuntelijametodi. Kuuntelijametodi käsittelee tulevat viestit niiden toimintoparametrin avulla. Toimintoparametri asetetaan, kun viestit lähetetään. Se kertoo verticlelle, millaisen toimenpiteen sen pitäisi suorittaa. Toiminnot erotellaan valintarakenteen avulla ja ohjataan niitä käsittäville metodeille.

EventBusin avulla verticlet keskustelevat toisille

Vertx-olion avulla käytetään EventBus-oliota. Eventbusin avulla lähetetään verticleille viestejä. Viestin vastaanottajaa ei tiedetä. Tiedetään ainoastaan lähetysosoite, joka voi olla mikä tahansa merkkijono. Viestille voidaan liittää pyyntöolio, joka voi olla tyypiltään JSON. Tähän olioon voidaan lisätä esimerkiksi HTTP-palvelimelle saapuvat parametrit. Lisäksi tulisi määrittää viestin toiminta, joka kertoo vastaanottajalle, millaisen toimenpiteen sen tulisi suorittaa. EventBus odottaa verticlen vastausta ja ohjaa automaattisesti vastaukset lähettäjälle.

4 Yhteenveto

Opinnäytetyön tekemistä aloitettiin ensimmäisellä tavoitteella eli tutkimalla projektissa käytetyt teknologiat. Tuotiin esiin niiden hyvät ja huonot puolet sekä tutustuttiin syvällisemmin niiden ominaisuuksiin ja rakenteisiin. Jotkut teknologiat, kuten Kotlin ja Vert.x ovat hyvin tuoreita, ja netistä tuen saaminen ongelmiin tuntui välillä hankalalta. Vaikeuksista huolimatta näiden kaikkien teknologioiden tutkiminen ja oppiminen olivat erittäin mielenkiitoista ja palkitsevaa.

Teknologioiden tutkimisen jälkeen opinnäytetyötä jatkettiin projektiosuudella. Projektin tarkoituksena oli toteuttaa Web-pohjainen monitorointi- ja analysointisovellus käyttäen tutkittuja teknologioita. Tavoitteena oli testata teknologiat käytännön tasolla. Toteutus aloitettiin soveltamalla ohjelmistotuotannon käytäntöjä. Määriteltiin sovelluksen toiminnallisuudet ja saavutettavat hyödyt. Kuvattiin sovelluksen vaatimukset ja rajoitteet. Käytiin läpi sidosryhmät ja käyttötapaukset.

Määrittelyn jälkeen suunniteltiin sovelluksen rakennetta. Selvitettiin miten asiakaspuolen ja palvelinpuolen ominaisuudet saadaan aikaiseksi. Suunniteltiin mm. sovelluksen arkkitehtuuri sekä pakkaus- ja komponenttikaaviot. Katsottiin komponenttien rakennetta ja niiden suhdetta muihin sovelluksen osiin.

Suunnitteluvaiheen jälkeen siirryttiin toteutusvaiheeseen, jossa toteutettiin sovelluksen komponentit vaatimusten mukaisesti. Koodiesimerkkien avulla selvennettiin sovelluksessa käytettyjen algoritmien rakennetta ja tarkoitusta. Katsottiin myös asiakaspuolen ja palvelinpuolen kehitysympäristöjen pystyttämistä ja tarvittavien teknologioiden asennusta.

Opinnäytetyön tuloksena saatiin vaatimusten mukaan toimiva sovellus, joka täyttää toimeksiantajan kaikki vaatimukset. Kuvassa 20 nähdään esimerkkidashboardi.

Kuva 20. Esimerkki dashboardi, jossa on kuusi diagrammia.

Kokonaisuutena kaikki tutkitut teknologiat osoittautuivat erittäin päteviksi Web-sovelluksen kehittämiseen. Angular 5 tarjoaa monia toiminnollisuuksia, jotka helpottavat asiakaspuolen kehitystä merkittävästi, vaikka sillä onkin jyrkkä oppimiskäyrä ja tuntuukin välillä monimutkaiselta.

Palvelinpuolen toteuttamiseen käytetyt teknologiat vaikuttivat lupaavilta. Kotlin- ja Vert.x-työkalupakki ovat molemmat tuoreita teknologioita ja niillä on paljon uusia ominaisuuksia, jotka voivat helpottaa palvelinpuolen kehitystä. Kuitenkin niiden käyttäminen yhdessä voi aiheuttaa alussa kokemattomalle kehittäjälle harmaita hiuksia, sillä ongelmatilanteessa tälle yhdistelmälle ei ole netistä saatavilla riittävästi apua. Vert.x-työkalupakkia kannattaa käyttää Javan kanssa, ellei kehittäjällä ole vahvaa Kotlin-osaamista, sillä monet esimerkit ja oppaat on kirjoitettu Javalla.

Lähteet

1 Angular Architecture Overview. 2017. Verkkodokumentti. Google. Https://angular.io/guide/architecture.

2 Is It Angular 2 Or Angular 4 Or Just Angular. 2017. Verkkodokumentti. Arunkumar Gudelli. Http://www.angularjswiki.com/tutorials/angular/is-it-angular-2-or-angular-4-or-just-angular/.

3 Features and benefits. 2017. Verkkodokumentti. https://angular.io/features.

4 The Good and the Bad of Angular Development. 2017. Verkkodokumentti. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/.

5 Keynote - STEPHEN FLUIN and IGOR MINAR. 2017. Video. Ng-conf. https://www.youtube.com/watch?v=anzsE2TbCyk.

6 The Advantages of TypeScript. 2017. Unna Mana. Verkkodokumentti. https://www.upwork.com/hiring/community/the-advantages-of-typescript/.

7 Angular 4 Tutorial for Beginners: Learn Angular 4 from Scratch. 2017. Video. Mosh Hamedani. https://www.youtube.com/watch?v=k5E2AVpwsko.

8 Vert.x nettisivu. 2018. Verkkodokumentti. http://vertx.io/.

9 Overview of the Vert.x event-driven architecture. Erich Onorez. 2015. Verkkodokumentti. https://erichonorez.wordpress.com/2015/12/04/overview-of-the-vert-x-event-driven-architecture/.

10 Inside Vert.x. Comparison with Node.js. Seongmin Woo. 2017. Verkkodokumentti. https://www.cubrid.org/blog/inside-vertx-comparison-with-nodejs.

11 A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers. 2018. Julien Ponge, Thomas Segismont & Julien Viet. Verkkodokumentti. http://vertx.io/docs/guide-for-java-devs/guide-for-java-devs.pdf.

12 3 reasons to choose vertx. Michael Heinrich. 2014. Verkkodokumentti. https://dzone.com/articles/3-reasons-choose-vertx.

13 Web framework benchmark. 2018. Verkkodokumentti. https://www.techempower.com/benchmarks/#section=data-r15&hw=ph&test=db.

14 What is Kotlin. Martin Heller. 2017. Verkkodokumentti. https://www.infoworld.com/article/3224868/java/what-is-kotlin-the-java-alternative-explained.html.

15 What is functional programming? Eric Elliott. 2017. Verkkodokumentti. https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0.

16 10 reasons why you should switch to Kotlin for programming Android apps. Manish Patel. 2017. Verkkodokumentti. https://jaxenter.com/10-reasons-switch-kotlin-android-apps-137631.html.

17 PrimeNg. Verkkosivu. 2018. https://www.primefaces.org/primeng/#/.

18 Chart.js. Verkkosivu. 2018. http://www.chartjs.org/.

19 IntelliJ. Verkkosivu. 2018. https://www.jetbrains.com/idea/.

OptiAnalyserin käyttötapaukset

Käyttötapaus	Työntekijöiden keräyshistoriadiagrammin luonti
Toimija	Varastopäällikkö
Esiehdot	Toimijalla on tunnukset ja oikeudet
Toimintakuvaus	1. Toimija kirjaudu OptiAnalyserille omilla tunnuksilla. 2. Hän valitse diagrammivalikolta ”luo keräyshistoria diagrammi” valikon. 3. Näytölle ilmestyy dialoginäkymä, jossa on parametreja diagrammin luontia varten. 4. Toimija valitse haluamansa parametrit ja painaa dialoginäkymän ”OK” nappulaa.
Lopputulos	Näkymälle ilmestyy diagrammi, joka sisältää parametrien avulla haettu dataa.
Huomiota	Jos varastopäällikkö antamilla parametreilla ei löydy tulosta, Näkymälle ilmestyy asiasta kertova ilmoitus.

Hyväksymiskriteerit:

- Toimija onnistuu luomaan diagrammin valitsemilla parametreilla.

Käyttötapaus	Varastotehtävän tilannediagrammin luonti
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimijalla on tunnukset ja oikeudet
Toimintakuvaus	1. Toimija kirjaudu OptiAnalyseriin omilla tunnuksilla. 2. Hän valitse valikolta ”luo varastotehtävien tilannediagrammi” valikon. 3. Näytölle ilmestyy dialoginäkymä, jossa on parametreja diagrammin luontia varten. 4. Toimija valitse haluamansa parametrit ja painaa dialoginäkymän ”OK” nappulaa.
Lopputulos	Näkymälle ilmestyy diagrammi, joka sisältää parametrien avulla haettu dataa.
Huomiota	Jos varastopäällikkö antamilla parametreilla ei löydy tulosta, Näkymälle ilmestyy asiasta kertova ilmoitus.

Hyväksymiskriteerit:

- Toimija onnistuu luomaan diagrammin valitsemilla parametreilla.

Käyttötapaus	Diagramminäkymän sijainnin muokkaus
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on luonut diagrammin
Toimintakuvaus	1. Toimija siirtää hiiren diagramminäkymän yläpuolelle. 2. Hiiren osoittama nuolikuvake muuttuu siirtymiskuvakkeeksi. 3. Toimija klikkaa ja siirtää dialoginäkymää pitämällä klikkausta. 4. Toimija vapauttaa klikkausta.
Lopputulos	Diagrammi siirtyy uuteen paikkaan sivulla.
Huomiota	Diagrammit voivat joutua päällekkäin.

Hyväksymiskriteerit:

- Toimija onnistuu muuttamaan diagrammin sijainti.

Käyttötapaus	Diagramminäkymän koon muokkaus
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on luonut diagrammin
Toimintakuvaus	1. Toimija siirtää hiiren diagrammin oikeaan alanurkkaan. 2. Hiiren osoittama nuolikuvake muuttuu kaksoisnuolikuvakkeeksi. 3. Toimija klikkaa ja siirtää hiirtä pitämällä klikkausta. 4. Näkymän koko muuttuu hiiren siirtämissuunnan mukaan. 5. Toimija vapauttaa klikkausta.
Lopputulos	Diagramminäkymän koko muuttuu.
Huomiota	Diagrammit voivat joutua päällekkäin koon muutoksen seurauksena.

Hyväksymiskriteerit:

- Toimija onnistuu muokkaamaan diagrammin kokoa.

Käyttötapaus	Diagrammin poistaminen
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on luonut diagrammin
Toimintakuvaus	1. Toimija klikkaa dialoginäkymän oikealla yläkulmalla olevaa poistamisnappulaa.
Lopputulos	Diagrammi poistuu dashboardilta.
Huomiota	Diagrammi ei poistuu kuitenkin selaimen muistilta, joten jos diagrammia halutaan poistaa lopullisesti niin on tallennettava dashboardia uudestaan poiston jälkeen.

Hyväksymiskriteerit:

- Toimija onnistuu poistamaan diagrammin.

Käyttötapaus	Diagrammin sisällön muokkaaminen
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on luonut diagrammin
Toimintakuvaus	1. Toimija klikkaa dialoginäkymän oikealla yläkulmalla olevaa muokkausnappulaa. 2. Näytölle ilmestyy diagrammin dialoginäkymä, jossa on valittu valmiiksi diagrammin luonti vaiheessa valitut parametrit. 3. Toimija muokkaa parametrit. 4. Toimija painaa OK nappulaa.
Lopputulos	Dialoginäkymä poistuu ja tilalle latautuu päivitetty diagramminäkymä.
Huomiota	Muokkausta voidaan peruttaa dialoginäkymän peruta nappulalla. Jos toimija haluaa säilyttää muokkaukset selaimen muistissa, hänen tulisi tallentaa dashboardia uudestaan diagrammin muokkauksen jälkeen.

Hyväksymiskriteerit:

- Toimija onnistuu muokkaamaan diagrammin sisältöä uusilla parametreilla.

Käyttötapaus	Dashboardin tallentaminen
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on luonut vähintään yhden diagrammin
Toimintakuvaus	1. Toimija klikkaa valikosta tallenna dashboard -nappulaa.
Lopputulos	Toimija saa ilmoituksen, että dashboardi on tallennettu.
Huomiota	Dashboardin tallentaminen tallentaa diagrammit selaimen muistiin, joten tämän jälkeen ei voi enää perua diagrammeille tehdyt muutokset.

Hyväksymiskriteerit:

- Toimija onnistuu tallentamaan dashboardinsa selaimen muistiin.

Käyttötapaus	Dashboardin poistaminen
Toimija	Varastopäällikkö, Muut käyttäjät
Esiehdot	Toimija on tallentanut dashboardin
Toimintakuvaus	1. Toimija klikkaa valikosta poista dashboardi -nappulaa.
Lopputulos	Toimija saa ilmoituksen, että dashboardi on poistettu.
Huomiota	Dashboardia ei voida enää palauttaa poistamisen jälkeen.

Hyväksymiskriteerit:

- Toimija onnistuu poistamaan dashboardinsa selaimen muistilta.